НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК _____

«До захисту допущено»

Завідувач кафедри СПСКС

_____ В.П.Тарасенко_
(підпис) (ініціали, прізвище)

"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
Комп'ютерні системи та компоненти

на тему: Багатомовний модель-орієнтований підхід до розробки веб-додатків

Виконав: студент II курсу, групи _КВ-61м_____
(шифр групи)

Ткаченко Роман Юрійович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник доц. каф. СПСКС, к.т.н., с.н.с. Боярінова Ю.Є. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Комп'ютерні системи та компоненти

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ _В.П.Тарасенко__
(підпис) (ініціали, прізвище)
«___»_____2018р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Ткаченко Роману Юрійовичу

1. Тема дисертації: багатомовний модель-орієнтований підхід до розробки веб-додатків, науковий керівник дисертації к.т.н., с.н.с. доцент Боярінова Юлія Євгенівна, затверджені наказом по університету від «22» березня 2018 р. №986-с

2. Термін подання студентом дисертації 11 травня 2018 р.

3. Об'єкт дослідження: застосування модель-орієнтованих підходів до розробки ПЗ для створення веб-додатків на основі систем керування вмістом.

4. Предмет дослідження: багатомовний модель-орієнтований підхід до розробки веб-додатків на основі систем керування вмістом.

5. Перелік завдань, які потрібно розробити:

- Розробити ефективний за певними показниками спосіб розробки веб-додатків, на основі систем керування вмістом;

- Дослідити модель-орієнтовані підходи розробки програмного забезпечення.

6. Перелік ілюстративного матеріалу: 30 рисунків, 2 таблиці.

7. Перелік публікацій

- Наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2018;

- Всеукраїнська науково-практична конференція студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем 2018».

8. Дата видачі завдання 5 вересня 2016 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
	Вивчення теоретичного матеріалу	15.08.2017	
	Розробка технічного завдання	10.01.2018	
	Аналіз існуючих рішень	05.02.2018	
	Розробка багатомовного підходу до розробки веб-додатків	05.03.2018	
	Аналіз результатів та розробка вказівок щодо створення мов моделювання	25.03.2018	
	Підготовка пояснювальної записки	14.04.2018	
	Попередній розгляд магістерської дисертації на кафедрі	26.04.2018	

Студент _____ Р.Ю.Ткаченко
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Ю.Є. Боярінова
(підпис) (ініціали, прізвище)

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API (англ. Application Programming Interface) - інтерфейс програмування додатків

CASE (англ. Computer-Aided Software Engineering) - набір інструментів і методів програмної інженерії для проектування програмного забезпечення

CMS (англ. Content Management System) - система керування вмістом

DSL (англ. Domain-specific language) - предметно-орієнтована мова програмування

DSM (англ. Domain Specific Modeling) - домен-орієнтоване моделювання

ECM (англ. enterprise content management) - управління документами та іншими типами контенту, а також їх зберігання, обробка та доставка в масштабах підприємства.

HTML (англ. HyperText Markup Language) - стандартна мова розмітки веб-сторінок в Інтернеті.

IL (англ. Intermediate Language) - проміжна мова

MDA (англ. Model Driven Architecture) - модельно-орієнтований підхід до розробки програмного забезпечення

MDE (англ. Model-Driven Engineering) - модель-орієнтована інженерія

ML (англ. Modeling Language) - мова моделювання

MOF (англ. Meta Object Facility) - стандарт мета-моделювання OMG

MYNK (англ. Model sYNchronization frameworK) - засіб для синхронізації моделей

OMG (англ. Object Management Group) - некомерційна міжнародна організація. Відповідає за розробку та затвердження незалежних IT-стандартів об'єктно-орієнтованого програмування.

SaaS (англ. Software as a service, SaaS) – програма, як послуга

UML (англ. Unified Modeling Language) - уніфікована мова моделювання

URL (англ. Uniform Resource Locator) - уніфікований локатор ресурсів або адреса ресурсу

XMI (англ. XML Metadata Interchange) - стандарт консорціуму OMG, для обміну ме́та-інформацією за допомогою та на основі XML

XML (англ. Extensible Markup Language) - стандарт побудови мов розмітки ієрархічно структурованих даних

ПЗ - програмне забезпечення

ВСТУП

З огляду на всебічну експансію Інтернету та всесвітньої мережі в останні роки зростає популярність веб-додатків. Системи управління контентом (англ. CMS), останнім часом набули особливого значення, оскільки вони полегшують розподіл широкого спектру контенту для користувачів. Хоча деякі з цих систем CMS дають можливість побудувати досить складні веб-додатки, розробка, як і раніше, здійснюється за допомогою традиційних методів, орієнтованих на вихідний код.

З появою мов високого рівня, що вважаються мовами третього покоління, сучасні способи розробки ПЗ почали змінюватись в сторону четвертого покоління, а саме, модель-орієнтованої інженерії (англ. MDE) . Дана парадигма спрямована на досягнення більш високого рівня абстракції та використання моделей у якості найважливіших об‘єктів у процесі розробки програмного забезпечення.

Сучасні процеси розробки ПЗ досі схильні до проблем, пов’язаних з неможливістю враховувати точки зору усіх учасників процесу, зазвичай відповідальність за об’єднання цих точок зору несе інженер-розробник. Це, в свою чергу, приводить до необхідності подальших змін або вдосконалень системи, тому що даний спосіб розробки ПЗ не гарантує, що розроблювана система відповідатиме очікуванням усіх учасників після першої ітерації розробки. Крім того, більшість сучасних модель-орієнтованих підходів до розробки ПЗ досі розглядають вихідний код у якості головного артефакту, тому що розробка моделей зазвичай вважається проміжним кроком для процесу отримання вихідного коду.

На відміну від інших підходів до розробки ПЗ, які, як правило, використовують одну мову програмування (C# , Java , PHP) та спрямовані на аудиторію технічних розробників, модель-орієнтований підхід зосереджений на вирішенні потреб як нетехнічних учасників процесу розробки ПЗ, так і

розробників. Для цього пропонується набір мов моделювання на різних рівнях абстракції та механізм синхронізації моделей різних рівнів.

Інтернет став потужною платформою для розгортання різних об‘єктів і систем, і також вплинув на спосіб, у який розроблюються додатки. Це в свою чергу призвело до появи множини веб-орієнтованих фреймворків та бібліотек, які дозволяють розробникам використовувати потужність веб-технологій для вирішення багатьох задач.

Результатом переходу з настільних додатків до веб-додатків стала поява багатьох веб-орієнтованих CMS та ECM систем, які забезпечують публікацію та управління контентом. Поява даних систем сприяла розвитку модель-орієнтованої інженерії , що має за мету замінити сучасні способи розробки ПЗ, використовуючи моделі у якості основних об‘єктів розробки, з яких генерується код та інші артефакти.

Модель-орієнтовані підходи до розробки ПЗ стають все більш популярними, завдяки розвитку модель-орієнтованої інженерії (MDE). MDE виступає за використання моделей, як основних об‘єктів у процесі розробки ПЗ, в свою чергу, документація та вихідний код, можуть бути отримані з цих моделей за допомогою автоматизованих перетворень. Завдяки даній методології, розробники можуть зосередити увагу на основних концепціях (замість вихідного коду та деталей реалізації), що у подальшому дозволяє ефективно керувати постійно зростаючою складністю ПЗ. Окрім автоматизації багатьох повторюваних задач, MDE також надає додаткові переваги , такі, як: зниження складності платформи або нездатність мов програмування виразити концепції домену; розгортання на декількох різних платформах без необхідності використання різного вихідного коду.

Хоча існують приклади реалізації MDE, наприклад, OMG MDA , більшість розробників досі використовують MDE у контексті домен-орієнтованого моделювання (англ. DSM) , у якому графічна мова орієнтована

на конкретну область застування, а розроблені моделі використовуються для генерації відповідного вихідного коду. Таким чином, сам процес розробки програмного забезпечення залишається орієнтованим на вихідний код, хоча деякі його частини спрощені завдяки DSL.

Підходи до розробки ПЗ на основі можна використовувати не тільки для створення веб-сайтів, але і як засіб для розгортання спеціалізованих веб-додатків. Системи CMS (які самі є веб-додатками) можуть ефективно підтримувати динамічне керування веб-сайтами та їх вмістом, і пропонують такі аспекти, як розширюваність і модульність, незалежність контенту і його презентації, підтримка багатьох типів контенту, підтримка управління доступом і користувачами, підтримка управління та виконання робочих процесів (англ. workflow).

Деякі системи CMS розвинулись до рівня фреймворків, тобто стали надавати засоби для створення веб-додатків з більш складним призначенням, ніж просте управління контентом. Також більшість систем управління контентом сприяють створенню складних веб-додатків за рахунок підтримання деяких високорівневих концептів, таких, як користувач, роль або модуль. Проте розробка та розгортання веб-додатків на базі CMS, як і раніше, здіснюється типовими засобами. Така розробка передбачає (1) написання вихідного коду, який використовує API, доступний CMS, використовуючи мову програмування, яка підтримується базовою технологією веб-сервера CMS (наприклад, C#, Java або PHP), компіляцію цього вихідного коду і розгортання об‘єктів в CMS, використовуючи механізм, вбудований в CMS або копіювання цих об‘єктів вручну на відповідний веб-сервер.

1 МЕТОДИ СТВОРЕННЯ МОВ МОДЕЛЮВАННЯ

1.1 Моделювання та модель-орієнтована інженерія

Створення моделі деякого феномену реального світу, здається, є природнім людським прагненням розуміти та прогнозувати явища в світі . Багато дисциплін використовує моделі, щоб спростити складні поняття, з метою кращого розуміння їх суті.

Моделювання завжди було важливим механізмом для подолання складності реальності. Хоча в науці моделі використовуються для опису існуючих явищ реального світу, в техніці моделі використовуються для опису систем, які будуть розроблятися в майбутньому. Таким чином, інженерні моделі зазвичай є конструктивними, тоді як наукові моделі є описовими.

Вчені управляють складністю явищ, які вони вивчають через моделювання. Звичайно, щоб бути корисними у якості засобу інформації, моделі повинні бути чітко виражені, тобто усі припущення повинні бути чітко висловлені та повідомлені мовою, яку можуть розуміти більшість зацікавлених науковців. Стаховіак дає детальне визначення моделювання , де три основні характеристики моделі описуються наступним чином:

1) існує оригінал,

2) модель є абстракцією оригіналу,

3) модель відповідає цілям оригіналу.

Наукові моделі, як правило, використовуються для розуміння реального світу та передбачення деяких його аспектів: за допомогою гравітаційних законів Ньютона ми можемо передбачити, скільки часу потрібно для того, щоб яблуко впало з дерева. Філософ К. Поппер характеризував наукові теорії як фальсифіковані моделі, тобто моделі, які можна порівняти з деякою спостережуваною реальністю, щоб оцінити, де вони знайдуть застосування. Таким чином, оригінал є частиною цього світу. Вчені абстрагуються від

складних деталей, і, як правило, моделі, які вони будують, застосовуються в межах певних кордонів, які також повинні бути явно визначені.

Деякі відомі моделі є неправильними, але все-таки вони досить добре пояснюють певні явища, наприклад, геоцентричні моделі Сонячної системи Кеплера. Абстракція завжди означає, що деякі властивості втрачаються, хоча, з огляду на мету моделі, достатньо детально відображені для виконання цілей моделі. Тому, на питання «Чи корисна модель?», можна відповісти лише знаючи її призначення.

Інженери (в тому числі інженери програмного забезпечення), як і вчені, використовують моделі для спрощення складних явищ. Однак головна відмінність полягає в тому, що феномен (наприклад, двигун, процес, програмне забезпечення), який моделюється, як правило, не існує на момент побудови моделі, оскільки мета інженерів - побудувати цей феномен з моделі. Тобто модель виконує роль плану або проекту.

У інженерії типовою задачею є розбиття складної системи на стільки моделей, скільки потрібно, щоб відповідати всім поставленим задачам таким чином, щоб вони піддавалися аналізу, стали зрозумілі та готові до побудови. Даний підхід поділу задач на більш прості витримав випробування часом. В галузі розробки програмного забезпечення, модель-орієнтована інженерія (англ. MDE) спрямована на зменшення випадкової складності, пов'язаної з розробкою складних програмних систем . Первинним джерелом випадкової складності є великий розрив між високорівневими поняттями, що використовуються експертами областей для вираження їх конкретних потреб та абстракцій низького рівня, що надаються мовами загального програмування. Усунення цього розриву вручну вимагає багато часу і зусиль. Підходи MDE вирішують цю проблему за допомогою методів моделювання, які підтримують розділ функцій та автоматичне створення основних об‘єктів моделей. У MDE модель описує аспект системи. Розділ функцій залежить від використання

різних мов моделювання, кожна з яких надає конструкції на основі абстракцій, специфічних для конкретного аспекта системи. Використання концепцій, специфічних для конкретних доменів та досвіду розробки на основі MDE, може суттєво підвищити продуктивність та якість розроблюваних систем.

Щоб покращити спосіб, у який розробляється програмне забезпечення, важливо, щоб усі учасники процесу могли виразити свої знання, наміри та задачі, використовуючи відповідні мови програмування. Дані мови можуть бути графічні (наприклад UML) або текстові (C#, C++, Python) в залежності від деяких факторів, таких як роль учасника, або рівень технічного знання.

Потреба в різноманітних мовах, в свою чергу, породжує потребу для механізмів створення цих мов, які визначатимуть нові шляхи для створення та визначення моделей. Більш того, питання створення нових мов з більшим рівнем абстракції має велику практичну цінність для учасників процесу, що виконують аналіз, будують архітектуру та розроблюють ПЗ.

В ході даного дослідження було розглянуто та проаналізовано дані теми та підходи:

1. модель-орієнтована інженерія;
2. модель-орієнтована архітектура;
3. домен-орієнтована інженерія;
4. метамоделювання.

Програмні системи сьогодні досягають такого рівня складності, що навіть мови програмування третього покоління, такі як Java чи C#, стають все менш спроможними їх підтримувати . Надмірна орієнтованість на те, як програмне рішення повинне працювати, а не на те, яким програмне рішення повинно бути - одна з основних проблем цих мов. Потрібні механізми та техніки, які дозволили б розробнику абстрагуватися від самої мови програмування і, натомість, сконцентруватися на створенні простих (і елегантних, наскільки це можливо) рішеннях певних проблем.

Впродовж останніх кількох років значні дослідницькі зусилля спрямовуються на те, щоб підняти рівень абстакції до модель-орієнтованої інженерії. Так звана модель-орієнтована розробка (англ. MDD), яка наразі знаходиться у процесі становлення, базується на систематичному використанні таких моделей, як основи для специфікації вимог ПЗ . На відміну від попередніх парадигм (орієнтованих на первинний код), MDD робить модель найважливішим об'єктом розробки, в той час як первинний код та документацію можна отримати з моделей автоматично, що позбавляє розробників проблем, пов'язаних зі складністю платформи чи неспроможністю мов третього покоління виразити певні концепти.

MDE не є новою ідеєю. У 80-х та 90-х роках XX-го століття було запропоновано набір інструментів - CASE (англ. Computer-Aided Software engineerung) для автоматизованого проектування програмного забезпечення. Метою CASE було забезпечити розробників методами та інструментами для створення програмних систем та дозволити графічне проектування з використанням мов загального призначення. Таким чином, розробники мали би можливість робити різні завдання з проектуваннями, наприклад, верифікувати їх або трансформувати у код (і навпаки). Проте, CASE інструменти не змогли досягти поставленої мети у зв'язку з такими проблемами, як : 1) погана відповідність між мовами загального призначення та платформами, що зробило процес генерування коду складним для розуміння та підтримки; 2) неможливість масштабування, оскільки інструменти не підтримували одночасну розробку; 3) збереження центральної ролі первинного коду, оскільки модель вважалася лише елементом документації. Сьогодні шанси успіху впровадження MDE зростають у зв`язку з тим, що розробники усвідомлюють обмежену здатність мов третього покоління підтримувати високий рівень складності ПЗ, а також платформи пропонують більше засобів для розробки моделей.

На сьогодні існує значна кількість підходів, пов’язаних з MDE, серед яких можна виділити Модель-орієнтовану архітектуру (MDA) і Домен-специфічне моделювання (DSM). Важливо наголосити на тому, що MOI не слід ототожнювати з конкретним методом реалізації. Натомість, MDE є парадигмою, яка визначає ряд підходів і є незалежною від мови програмування чи технології .

Всі підходи, пов’язані з MDE, мають спільні базові поняття. Модель є інтерпретацією певного домену - схематичним зображенням фрагменту реального світу, на який спрямовані завдання моделювання та розробки. Іншими словами, модель можна розцінювати, як стиснуте зображення системи, яке містить характеристики, важливі для конкретної перспективи (точки зору) .

На більш абстрактному рівні організаційну стуктуру моделі визначає метамодель. Метамодель – це набір правил і концептів, які описують можливу структуру моделі і визначають мову моделювання, за допомогою якої буде створено модель. Подібно до того, як метамодель визначає мову моделювання, метаметамодель визначає мову, за допомогою якої буде створено метамодель (рис. 1). Рівні абстракції, описанні вище, утворюють ієрархічну структуру метарівнів. Поняття метарівня важливе для створення архітектури в модель-орієнтованій архітектурі.

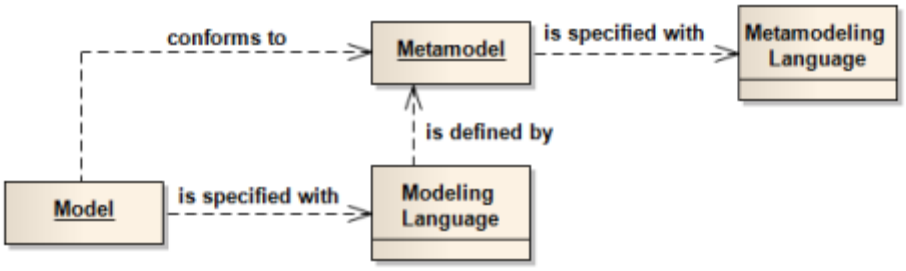

Рисунок 1 - Зв'язок між моделлю та метамоделлю

Більшість інструментів моделювання, які створено за допомогою об’єктно-орієнтованих мов програмування моделювання, використовують

один метарівень, який базується на відношенні клас-сутність, що дозволяє підтримувати тільки один логічний рівень.

1.1 Підхід OMG

OMG визначила широко відомий підхід, що базується на парадигмі MDE і називається модель-орієнтована архитектура (MDA). Даний підхід базується на стандартах OMG, які використовують специфікацію мов моделювання та трансформації моделей. Деякі з цих стандартів (MOF та UML) пропонують засоби для створення нових мов моделювання.

1.1.1 Уніфікована мова моделювання (UML)

Уніфікована мова моделювання (UML) є мовою моделювання загального призначення. UML була створена для визначення, візуалізації і документування ПЗ, але також використовується рядом засобів, методологій та досліджень у відношенні розробки ПЗ та генерації вихідного коду.

Нинішня версія UML включає в себе 14 різних видів діаграм (рис. 2), розділених на дві категорії:

1. структурні діаграми, які використовуються для відображення статичних структур і об’єктів в модельованій системі;
2. діаграми поведінки, що використвуються для зображення динамічних аспектів і поведінки об’єктів у системі.

UML визначає такі типи структурних діаграм: класів, компонент, композитної/складеної структури, кооперації, розгортування, об'єктів, пакетів; та діаграми поведінки: діяльності, станів, прецедентів, кооперації (UML1.x) / комунікації (UML2.0), огляду взаємодії, послідовності, синхронізації.

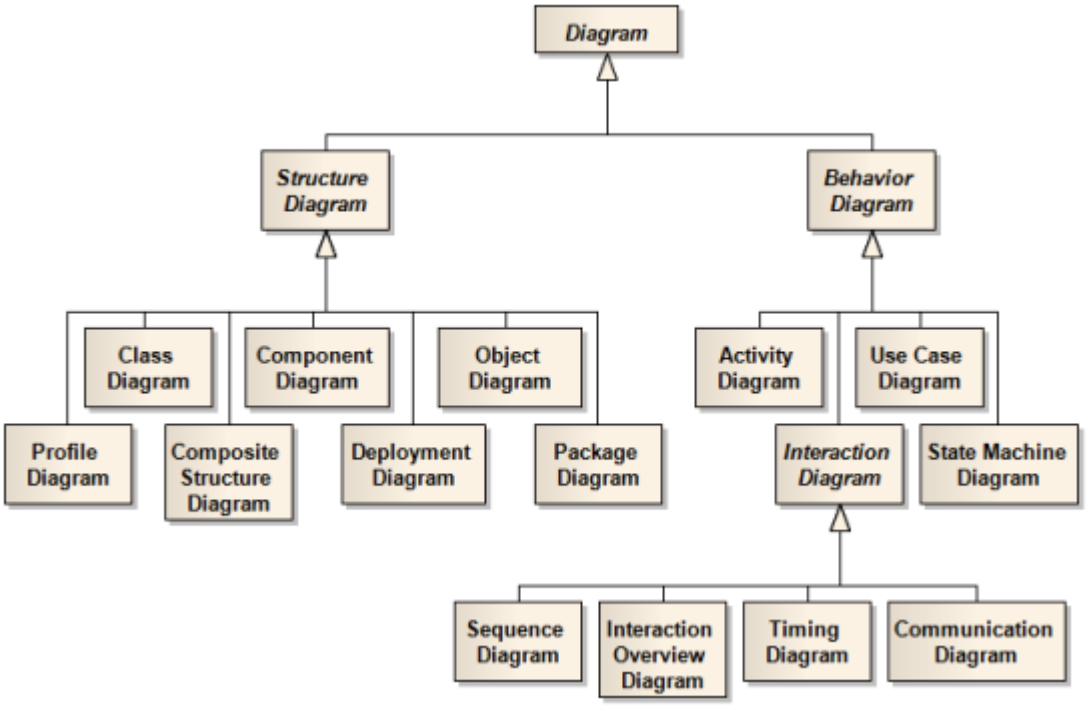

Рисунок - 2 Різні види UML діаграм

Хоча UML була кроком вперед у врегулюванні стандарту, що розуміється цілою спільнотою розробки ПЗ, вона все ще критикується через такі причини, як:

1. будучи легким для використання в специфічних для програмного забезпечення областях, як наприклад IT або системи телезв'язку, UML досить важко застосувати в істотно інших областях, таких як біологія або фінанси ;
2. UML не орієнтовано на використання на практиці ;
3. занадто велика складність.

UML традиційно використовувалась у якості метамоделі (тобто розробники створюють моделі, використовуючи мову, що була створена за допомогою UML). Однак, специфікація UML також пропонує механізм профілювання, який дозволяє визначати нові позначення або номенклатури, надаючи спосіб розширення метакласів UML і їх адаптації для різних призначень.

Профілі можна розглядати як збірку стереотипів, спеціальних значень та обмежень. Стереотип визначає додаткові властивості елемента, але ці властивості не повинні суперечити властивостям, які вже пов'язані з початковим елементом моделі. Таким чином, профіль не дозволяє користувачеві редагувати існуючу UML метамодель, але дозволяє її розширення, що орієнтує модель на якийсь конкретний домен.

1.1.2 Мета-об’єктний засіб (MOF)

Мета-об’єктний засіб (MOF) разом з UML є основою підходу OMG до модель-орієнтованої інженерії. UML та MOF було спроектовано так, щоб вони самі були сутностями MOF. Це було досягнуто за рахунок розробки UML інфраструктурної бібліотеки, яка надає засоби для моделювання структури UML та MOF та також може бути використана для інших метамоделей. На рис. 3 показані залежності між UML і MOF. Зверніть увагу, що MOF може описати сам себе, що робить його рефлексивним. Крім UML, OMG також визначила інші стандарти на основі MOF, такі як метадані XMI та QVT.

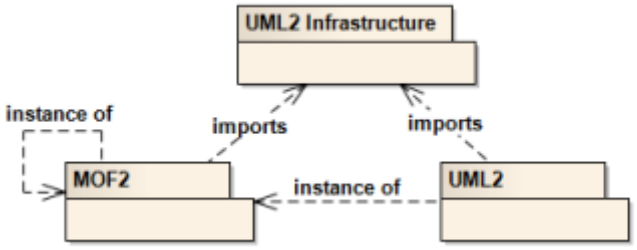

Рисунок - 3 Залежності між UML та MOF

Коли використовують термін MOF, фактично мається на увазі один з двох варіантів метамоделей MOF, EMOF або CMOF. EMOF (Essential MOF) - це підмножина MOF, яка особливо підходить для використання в підходах до об'єктно-орієнтованого моделювання та XML. Таким чином, EMOF дозволяє відображати моделі MOF у документах XML (наприклад, за допомогою формату XMI, представленого далі в цьому розділі) або навіть у вигляді

концепцій об'єктно-орієнтованого програмування. На рис.4 наведено огляд можливостей, визначених у контексті EMOF.

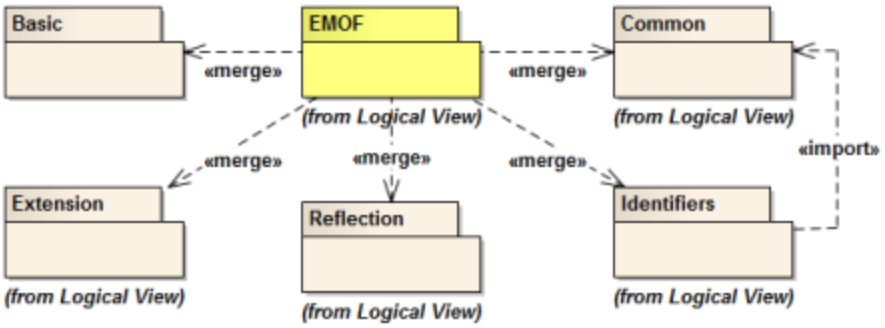

Рисунок - 4 Огляд можливостей у контексті EMOF

З іншого боку, CMOF (Complete MOF) – це те, що, як правило, мається на увазі, коли йдеться мова про MOF. CMOF є результатом об'єднання (або об'єднання через оператор злиття пакетів) EMOF з його розширеннями, які передбачають перевизначення елементів, які роблять CMOF достатнім для базових потреб метамоделювання. На рис. 5 зображено CMOF як комбінацію EMOF з цими додатковими можливостями.

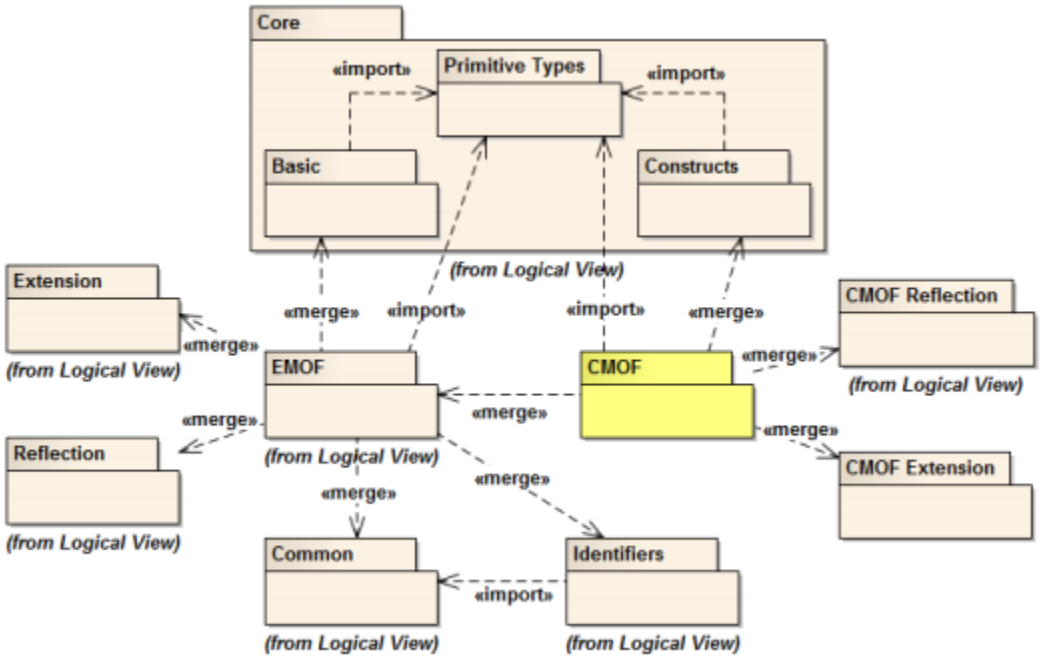

Рисунок - 5 CMOF як комбінація EMOF

Обмін XML метаданими (XMI) дозволяє обмінюватися будь-якими метаданими, метамоделі яких можна визначити в MOF. Це дозволяє здійснювати відображення будь-якої метамоделі на основі MOF в XML, надаючи ефективний спосіб серіалізації та обміну моделями між різними інструментами. Тим не менш, користувачі часто вважають XMI неефективним засобом для обміну моделями, оскільки інструменти часто використовують власні розширення XMI для конкретного постачальника, потенційно втрачаючи інформацію при обміні моделями між інструментами різних постачальників.

З іншого боку, специфікація QVT визначає спосіб перетворення вихідних моделей у цільові моделі, дозволяючи визначати операції запиту, перегляду та перетворення над моделями. Одна з найбільш цікавих ідей QVT полягає в тому, що саме перетворення є моделлю на основі MOF, що робить QVT сумісним з MOF.

Особливо важливим стандартом для мов на базі QVT та MOF є мова обмеження об'єктів (OCL). Хоча вона спочатку була тісно пов'язана з UML, її масштаби з того часу розширювалися, і OCL стала частиною підходу, що ґрунтується на OMG. OCL є текстовою декларативною мовою, яка використовується для визначення правил та запитів для моделей на основі MOF та метамоделей (включаючи UML), коли наявні діаграми операторів не мають достатньої виразності для визначення таких правил.

MOF був розроблений, щоб бути базою підходу OMG. OMG визначила чотири рівні архітектури метамоделі, яка базується в основному на MOF та UML, що не тільки дозволяє користувачам визначати власні UML-моделі, але також дає змогу визначати інші мови на основі MOF, такі як CWM. На рис. 6 показана така архітектура:

1. MOF - це метаметамодель, на M3 метарівні;
2. UML - екземпляр MOF - метамодель, на M2 метарівні;

3. модель користувача містить моделі, визначені за допомогою UML, на M1 метарівні;
4. M0 метарівень містить поточні екземпляри моделей, визначених на M1 метарівні.

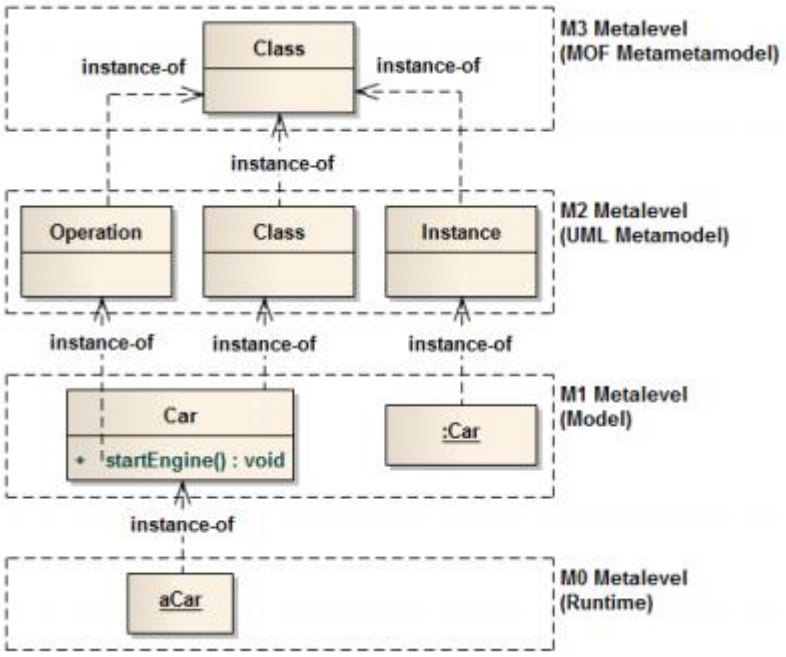

Рисунок - 6 Чотирьох-рівнева архітектура метамоделей

Як показує малюнок, MOF відіграє роль метаметамоделі (тобто забезпечує мову метамоделювання). Таким чином, концепції, визначені MOF (а саме - EMOF), не мають на меті визначати моделі програмних систем, а надають засоби для визначення нових концепцій в метамоделі, тим самим створюючи нову мову моделювання.

1.1.2 Модель-орієнтована архітектура

Модельно-орієнтована архітектура (англ. MDA) – це підхід до циклу розробки програмного забезпечення, в основі якого лежить моделювання системи. Більша увага в цьому підході надається трансформації моделі, ніж, власне, метамоделюванню.

MDA визначає три типи моделі :

1. незалежна від обчислення модель;

2. незалежна від платформи модель;
3. специфічна для платформи модель.

Незалежна від обчислення модель (англ. CIM) – це модель, яка зображає системні вимоги, зокрема, визначає характеристики середовища, в якому система буде функціонувати та описує функції, які система повинна виконувати. З іншого боку, незалежна від платформи модель (англ. PIM) – це модель високого рівня абстракції, яка не залежить від технології імплементації. Це робить її зручною для опису системи з загальної точки зору без врахування деталей щодо технології та імплементації, таких як реляційні бази даних чи сервери застосунків. Нарешті, специфічна для платформи модель (англ. PSM) є також моделлю програмного забезпечення, але на відміну від PIM та CIM, вона конкретно визначає технологію імплементації.

PIM можна трансформувати в PSM (або ж кілька PSM, кожна з яких відповідає певній технології) шляхом трансформації моделі-у-модель. Можливість створення декількох PSM з однієї PIM зумовлена тим, що сучасне програмне забезпечення часто застосовує кілька технологій (наприклад, сервер для веб-додатку, який використовує JBoss у поєднанні з системою баз даних MySQL). Модель-орієнтована архітектура прописує існування певних правил для трасформації, але не визначає ці правила. У деяких випадках, вендор може забезпечувати правила, як частину стандартного комплекту моделей та профілів.

На рис. 7 подано огляд MDA (CIM вилучено для спрощення). Суцільні лінії, які з'єднують прямокутники — це трансформації, які визначаються правилами трансформації.

MDA зазнає критики з боку інженерів програмного забезпечення у зв'язку з тим, що: 1) підхід використовує мову UML, яку часто критикують (або сприймають неоднозначно, щоб не повторювати критика двічі) ; 2) якщо виникає проблема, розробник має вручну внести правки в автоматично

генерований код, що часто викликає труднощі; 3) ті, хто створює первинний код можуть генерувати значну частину додатку, але код, який вони отримують в результаті, часто вимагає значних зусиль для інтеграції зі специфічними для окремих платформ API (наприклад Java чи .NET).

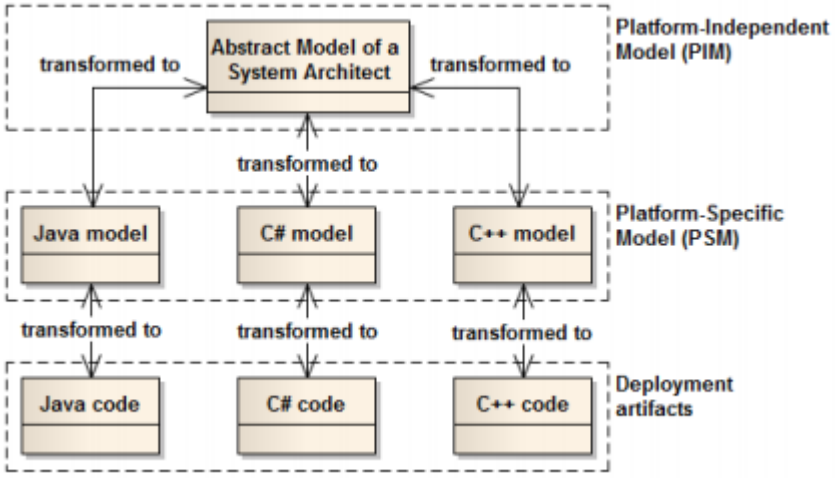

Рисунок - 7 Огляд підходу MDA

1.2 Домен-орієнтоване моделювання

Домен-орієнтоване моделювання (англ. DSM) використовує поняття з проблемного домену у якості базових будівельних блоків моделей, на відміну від традиційних інструментів CASE, які використовують поняття мов програмування (наприклад, клас, об'єкт).

З технологічної точки зору DSM підтримується інструментом DSM, який можна розглядати, як додаток для створення спеціальних інструментів CASE (або, іншими словами, середовища для створення інструментів CASE, які можуть бути використані для створення додатків). Таким чином, інструменти DSM додають абстрактний рівень над традиційним CASE, надаючи можливість специфічної для домену конфігурації отриманої моделі, як показано на рис. 8. Через цей рівень абстракції інструменти DSM також називаються інструментами метамоделювання.

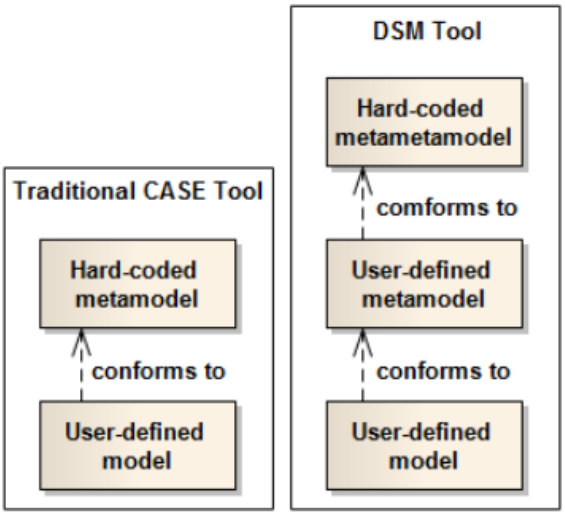

Рисунок - 8 Різниця між CASE та DSM

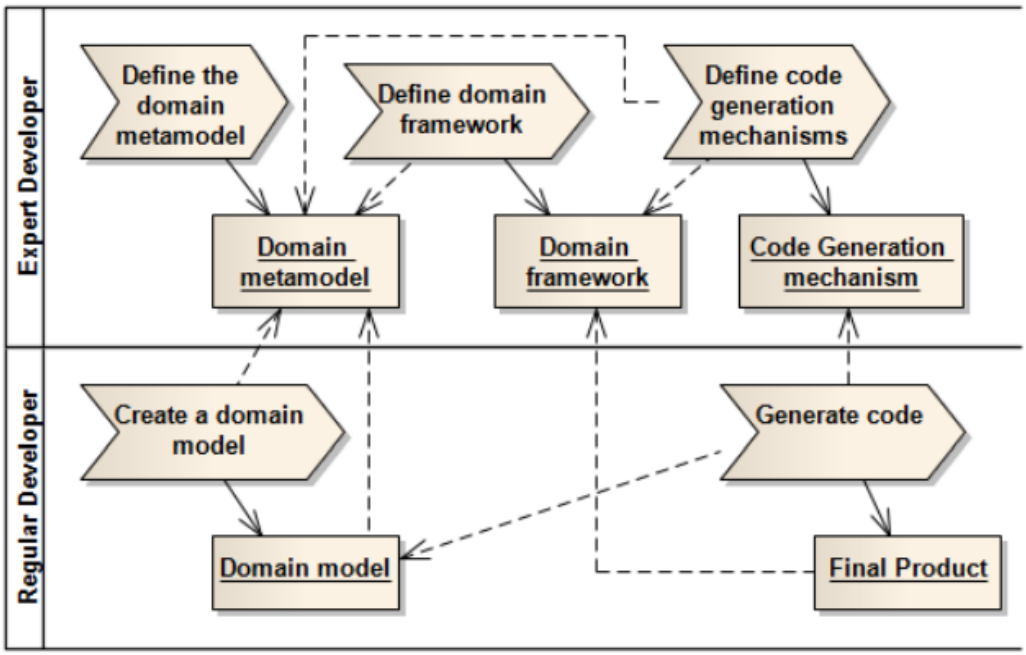

Рисунок - 9 Типовий DSM процес

DSM тісно пов'язана з концепцією Domain-Specific Language (DSL). DSL - це мова, призначена для вирішення завдання (або набору завдань) у певній проблемній області, на відміну від мови загального призначення . Як показує рис. 9 DSL та відповідні генератори зазвичай визначаються розробниками, які є експертами в проблемній області, через високоспеціалізований характер, що лежить в основі DSL. Інші розробники, менш досвідчені в відображенні

понятть домену за допомогою вихідного коду, після цього виконують свою роботу використовуючи DSL.

Відомим прикладом DSL є стандартна мова запитів SQL (англ. SQL Query Language), яка є стандартною текстовою мовою для доступу до баз даних (іншими словами, SQL - це текстовий DSL для проблемного домену запитів та маніпулювання базою даних).

Розробники зазвичай вважають за краще використовувати DSL, а не загальні мови моделювання, такі як UML, через концепції та абстракції, які надаються кожною DSL. Загальні мови використовують концепції, пов'язані з об'єктно-орієнтованими програмуваннями, які розміщують моделі на рівні абстракції трохи вище вихідного коду. З іншого боку, DSL призначені для використання концепцій з проблемного домену, що призводить до того, що розробники можуть абстрагуватися від того, як ці поняття відобразяться у вихідному коді, що в свою чергу, призведе до більшого фокусування на вирішенні проблеми, а не витрачання часу на розробку відповідності між рішенням проблеми та реалізацією цього рішення.

Тим не менше, навіть загальні мови моделювання іноді розглядаються як DSL, коли вони призначені для використання з певною метою. MOF часто розглядається як DSL, в якому метою мови є визначення нових мов моделювання. UML також може використовуватися для визначення DSL, зокрема, за допомогою механізму профілювання.

Важливим аспектом, який слід враховувати при визначенні DSL, є її якість (зокрема зручність використання). Існує значна кількість пов'язаних робіт щодо цієї теми, які висвітлюють деякі критерії, які можуть бути використані для оцінки якості DSL. До таких критеріїв відносяться:

1. Ефективність - з'ясування того, чи може розробник писати мовні конструкції без помилок;

2. Продуктивність - вимірювання кількості зусиль, які розробник повинен виконати для написання мовних конструкцій;
3. Задоволення – визначення того, чи стає розробник роздратованим при використанні мови;
4. Доступність, яка визначає, чи легко навчатися та запам'ятати мову.

1.3 Задачі подальших досліджень

Головні дослідницькі проблеми (та запропоновані рішення), що будуть розглянуті, можуть бути узагальнені у даний список:

1. Більшість існуючих модель-орієнтованих підходів до розробки базуються на одній мові моделювання;
2. Існуючі мови пропонують або (1) абстрагувати розробника від низькорівневих деталей розробки, або (2) позбутися вищезгаданної абстракції, та побудувати повний, повністю фунціональний та широко кастомізований додаток;
3. Результатом сучасніх модель-орієнтованих підходів до розробки веб-додатків є, як і раніше, вихідний код, замість моделей.

Підходи MDE засновані на одній мові. Перша проблема - сучасні модель-орієнтовані підходи до розробки ПЗ (орієнтовані на веб-додатки або інші) використовують одну мову моделювання: стандартну OMG UML або іншу мову, орієнтовану на конкретний підхід. Це не є проблемою, якщо розробник є єдиним учасником процесу розробки програмного забезпечення, бо мова моделювання буде орієнтована на вирішення завдань розробника.

Проте, зазвичай в процесі розробки програмного забезпечення також приймають участь інші зацікавлені сторони (а саме кінцеві користувачі програми), кожен з яких має певний погляд стосовно того, які задачі розроблювана система повинна вирішувати. Як правило, розробнику надається певний набір вимог, таких, як необхідні функції додатку або бізнес правила, і виникає задача відображення цих вимог та функцій у контексті

використовуваної мови реалізації ПЗ (тобто, мануальне встановлення відповідності між поставленою задачею та фінальним рішенням). На практиці, даний підхід не є ефективним, тому що результат сильно залежить від інтерпретації розробником поставлених вимог.

Мова моделювання або занадто проста, або занадто складна. Більшість мов моделювання, які намагаються абстрагувати дизайнера від низькорівневих деталей реалізації, наприклад, WebML або XIS2, зазвичай не достатньо виразні для розробки повнофункціонального додатка, і навпаки. Хоча це має сенс (оскільки модель повинна бути абстракцією над реальністю), але врешті-решт призводить до ручного редагування вихідного коду – практики, якої модель-орієнтована інженерія (MDE) має намір уникнути. З іншого боку, підходи, які є дуже виразними, такі, як мова OutSystems Agile Platform, ймовірно, не дають реальної абстракції від деталей реалізації (що є одним із головних принципів MDE), але лише надають інший синтаксис для тих самих концепцій.

Ця проблема пов'язана з попередньою, оскільки ці мови зазвичай є результатом спроб сконцентрувати занадто багато деталей різних рівнів абстракції - наприклад, високорівневих моделей та деталей реалізації - в одній мові. Це, в свою чергу, робить мову або дуже складною, але здатну моделювати найбільш повні додатки, або відносно простою, але без можливості моделювати реальні програми.

Вихідний код залишається основним артефактом. Хоча мови програмування самі могли б розглядатися як модель-орієнтовані мови, вони все-таки орієнтовані на те, як комп'ютер повинен вирішити проблему, а не яка проблема повинна бути вирішена. Крім того, хоча більшість підходів дозволяють розробникам безпосередньо редагувати згенеровані артефакти низького рівня, одна з головних цілей MDE полягає саме в тому, щоб уникнути цього редагування. Більш того, якщо процес створення вихідного коду буде

запущений знову, то, ймовірно, що ручні зміни, внесені до вихідного коду, будуть втрачені.

Знову ж таки, ця проблема пов'язана з попередньою, оскільки відсутність виразності у більшості мов моделювання, ймовірно, випливає з того, що їх було створено з припущенням, що моделювання буде здійснюватися, як проміжний процес розробки, а не як процес створення основного артефакту, який буде розгорнуто у кінцевому середовищі.

Це ще одне питання, чим відрізняється підхід "OutSystems". Хоча підхід досі використовує генерацію та компіляцію вихідного коду, ці кроки виконуються "за кадром" і розробник не може втрутитися в цей процес. Це, у свою чергу, змушує мову моделювання бути достатньо виразною для задоволення потреб низького рівня для розробки більшості типів веб-додатків.

Метою даної роботи є дослідження модель-орієнтованих підходів розробки ПЗ, а також створення нового способу розробки веб-додатків, на основі систем управління вмістом, який буде більш ефективним за певними показниками.

Наукова задача, що розв’язується в даній роботі, включає наступні питання:

1. Як модель-орієнтовані підходи можуть ефективно вирішувати проблеми усіх учасників розробки ПЗ?
2. Чи може мова моделювання одночасно вирішувати проблеми на різних рівнях абстракції? Чи дійсно необхідно мати одну мову, що буде компромісом між підтримкою низькорівневих деталями та простотою вивчення і використання?
3. Чи буде підхід, що використовує декілька рівнів абстракції, ефективнішим за традиційних підхід до розробки ПЗ?
4. Чи можливо використовувати моделі у якості основного артефакта?

У цьому розділі було представлено парадигму MDE та деякі важливі підходи до формування мови моделювання, які є актуальними для цієї роботи, а саме, підхід розробки програмного забезпечення MDA (який базується на застосуванні перетворень моделей та таких стандартів, як UML, або мов моделювання MOF) та підхід DSM (який підтримується визначенням та використанням DSL).

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ РОЗРОБКИ ВЕБ-ДОДАТКІВ

2.1. Модель-орієнтовані підходи до розробки веб-додатків

Розробка веб-додатків у поєднанні з новими підходами щодо масштабування та розповсюдження додатків (такими як «Програма як послуга» (англ. Software as a service, SaaS)) для якнайбільшого числа користувачів, стала стандартним способом створення нових додатків (або для оновлення старих). В даний час існують деякі модель-орієнтовані підходи до розробки веб-додатків. Однак підхід до розробки зазвичай залежить від виразності мови моделювання, яка використовується в такому підході. У цьому розділі ми аналізуємо такі підходи MDE для розробки веб-додатків та їх мови:

1. WebML1 ;
2. UWE2 ;
3. XIS2;
4. Microsoft Sketchflow;
5. OutSystems.

Методи WebML, UWE та OutSystems були вибрані для аналізу (замість інших можливих підходів MDE), оскільки вони добре відомі в області розробки веб-додатків. Хоча Microsoft Sketchflow не є підходом MDE або мовою, і не орієнтований спеціально на веб-додатки, його було включено в цей аналіз, оскільки він може використовуватися для створення міжплатформових інтерактивних веб-програм, а також тому, що він розглядає дуже важливий аспект, який не розглядається в інших підходах, - надання зручного для користувачів бачення дизайну прикладних програм.

2.2. Порівняльний аналіз

У цьому розділі представлено аналіз вищезазначених підходів до розробки веб-додатків: (1) спочатку вводяться критерії аналізу, а саме, аспекти,

що будуть розглядаться; (2) результати цього аналізу, а також обгрунтування отриманих значень.

2.2.1. Критерії аналізу

Головним чином, аналіз зосереджений на мові моделювання, що використовується підходом, і чи стосується вона ряду проблем моделювання, таких як моделювання доменів, моделювання бізнес-логіки та моделювання користувальницького інтерфейсу. Тим не менш, також аналізуються деякі аспекти, пов'язані з генерацією частин веб-додатків, наприклад, використання перетворень моделі до моделі або розгортання у середовищі.

1. Моделювання домену. Моделювання домену стосується ідентифікації концепцій проблемної області та їх представлення, використовуючи мову моделювання (наприклад, UML-діаграми). Цей аспект аналізується з огляду на: (1) чи він підтримується мовами; і (2) чи є він незалежний від інтерфейсу користувача.
2. Моделювання бізнес-логіки. Хоча визначення моделювання бізнес-логіки можна вважати дещо суб'єктивним, в даному дипломі це розглядається, як специфікація поведінки веб-додатка. Цей аспект аналізується з огляду на: (1) чи підтримує він запити та маніпулювання поняттями домену (за допомогою використання шаблонів); (2) чи підтримує запити та маніпулювання низькорівневим способом, подібним до традиційного вихідного коду; (3) чи підтримує специфікацію процесу. Слід зазначити, що підтримка низького рівня вважається актуальною, оскільки вона часто відображає виразність мови: типові код-орієнтовані мови (такі як C або Java), хоча і є складними, але дуже виразні.
3. Моделювання навігації. Підтримка визначення потоку навігації (в контексті модельованого веб-додатку) між різними HTML-

сторінками або навіть всередині HTML-сторінок також є аспектом для аналізу.

4. Моделювання інтерфейсу користувача. Інший важливий аспект - підтримка моделювання користувальницького інтерфейсу. Проаналізовані теми: (1) чи мова моделювання користувальницького інтерфейсу не залежить від платформи (тобто не вимагає спеціального програмного забезпечення для представлення інтерфейсу користувача); (2) підтримує специфікацію керування доступом (тобто певні елементи керування відображаються або приховуються відповідно до аутентифікованого користувача); (3) дозволяє визначати користувацькі елементи інтерфейсу; (4) дозволяє використовувати шаблони взаємодії (наприклад, створювати, редагувати або асоціювати та відокремити); і (5) підтримує зв'язування між елементами інтерфейсу користувача та елементами моделі домену.
5. Перетворення моделі в модель. Цей аспект показує, чи використовує аналізований підхід перетворення моделі в модель. Такі перетворення, як правило, використовуються для прискорення проектування веб-додатків за допомогою аналізу моделі та механізмів виводу для автоматичного визначення деяких частин моделей веб-додатків, тим самим звільняючи розробника моделі від деяких повторюваних завдань.
6. Згенерований додаток є повним. Цей аспект визначає, чи має інструмент підходу можливість повністю генерувати програму (тобто, вона не вимагає ручної доробки специфічних функцій програмістами).
7. Незалежність від середовища розгортання. В цьому аспекті вивчається, на які цільові платформи націлений розглянутий

підхід, а саме, чи існує тісний зв'язок між підходом та цільовою платформою.

2.2.2. Результати аналізу

Аналіз цих підходів до розробки веб-додатків, відповідно до критеріїв аналізу, перерахованих у попередньому підрозділі, дає результати, наведені в табл. 1.

Таблиця 1 - Характеристики проаналізованих модель-орієнтованих підходів та мов

	Web-ML	**UWE**	**XIS2**	**Out-systems**	**Sketch-flow**
1. Моделювання домену	+	+	+	+	-
Незалежність від користувацького інтерфейсу	+	-	+	+	-
2.Моделювання бізнес-логіки	+	+	+	+	+
Маніпуляція доменом за допомогою шаблонів	+	-	+	+	-
Власні шаблони	+	-	+	-	-
Низькорівнева розробка	+	+	-	+	+
Доменний запит	+	+	-	+	+
Маніпуляція доменом	+	-	-	+	-
Визначення процесів	-	+	-	+	-
3. Моделювання навігації	+	+	+	+	+
4**.Моделювання інтерфейсу користувача**	+	+	+	+	+
Керування доступом	+	-	+	+	-
Власні елементи інтерфейсу	+	-	-	+	+
Шаблони взаємодії	+	-	+	+	+

Зв’язок інтерфейсу та елементів домену	+	+	+	+	+
5.Перетворення моделі до моделі	-	+	+	-	-
6. Згенерований додаток є повним	-	-	-	+	-
7. Незалежність від середовища розгортання	+	+	+	-	+

Моделювання домену є важливим аспектом у будь-якому підході розробки додатків. Проаналізовані підходи дозволяють забезпечити незалежність між моделлю домену та іншими частинами додатків, але лише WebML та XIS2 дозволяють моделювати домен таким чином, що модель домену повністю не залежить від решти деталей програми, а також не вимагає коректировки за допомогою інтерфейсу користувача.

У платформі Outsystems модель домена моделюється, як схема бази даних, хоча використовує інші терміни (об'єкти та атрибути, а не таблиці та стовпці); хоча ця недостатня абстракція може здаватися недоліком, на практиці вона наділяє розробника більшим рівнем контролю над реальною схемою бази даних веб-додатків. WebML також передбачає спосіб налаштування моделі домену для особливих потреб інших шарів за допомогою моделі похідних, а XIS2 забезпечує представлення BusinessEntities для обробки об'єктів домену іншими шарами.

Моделювання бізнес-логіки пропонується всіма аналізованими підходами, хоча і дуже різними способами. XIS2 розглядає цей аспект за допомогою Business-Entities - дозволяє іншим шарам додатку маніпулювати елементами, які є більш абстрактними, ніж елементи домену, та шаблонів (підтримуються типові операції створення, читання, оновлення, видалення, а розробник може реалізувати більше операцій, хоча вони повинні бути впроваджені вручну). WebML використовує аналогічні механізми (модель

походження (англ. Derivation Model)) для коригування моделі домену для інших шарів, блоків вмісту (англ. Content Units) та операцій (англ. Operation Units) для запиту та керування доменом. Розробник може додати нові операції, які повинні бути введені вручну, а також організувати їх у вигляді блок-схеми, що дозволяє визначити специфікації робочих процесів маніпулювання даними. UWE розглядає цей аспект лише за допомогою діаграм класу Process Model, які представляють взаємозв'язок між різними процесами та діаграм активності, які визначають етапи кожного процесу; сама реалізація повинна виконуватися вручну. Sketchflow, хоча і є тільки інструментом створення прототипів, дозволяє дизайнерам наповнити свої прототипи певною поведінкою за допомогою механізму "Поведінка"; вона також може бути розширена додатковими моделями поведінки, але створення таких функцій передбачає певну ручну розробку.

Нарешті, платформа Outsystems дозволяє розробникам графічно визначати складну бізнес-логіку: за допомогою визначення дій, таких як запити та шаблони маніпуляцій, розробники можуть визначити більшість функцій, які зазвичай повинні бути закодовані вручну.

Моделювання навігації розглядається всіма аналізованими підходами, оскільки це є основою для будь-якого веб-додатка. Завдяки спільній структурі веб-додатків (HTML-сторінки, гіперпосилання для переміщення між сторінками, використання параметрів запиту або аналогічного механізму для надання необхідних параметрів), всі підходи дотримуються однакових правил: малюється орієнтований граф, де вузли відповідають HTML-сторінкам, а ребра відповідають тим потокам навігації, які доступні користувачеві в контексті певної сторінки. Тим не менше, аналізовані підходи різняться в тому, чи може розробник визначити набори ребер (тобто дій або посилань), доступних в кожному вузлі. В WebML кожен вузол DataUnit має визначений набір посилань для введення та виведення, а розробник не може вказати додаткові посилання.

XIS2 дозволяє конструктору визначати дії, пов'язані з елементами користувальницького інтерфейсу на сторінці, а потоки навігації потім будуть асоціюватися з цими діями сторінки. Підхід XIS2 дуже схожий на ті, які можна знайти в платформі Outsystems, Sketchflow та UWE.

Моделювання інтерфейсу користувача. За винятком WebML, всі аналізовані підходи моделюють інтерфейс користувача графічним WYSIWYG (ви бачите те, що отримуєте) способом. WebML лише дозволяє вказати, які елементи будуть присутні на сторінці, але не їх розташування. Також платформа Outsystems, WebML та Sketchflow підтримують створення нових елементів інтерфейсу або сторінок для повторного використання в інших сторінках або додатках. Це дозволяє розробникам визначати певні розділи сторінки лише один раз та імпортувати ці розділи на інші сторінки додатка, таким чином, зміни такого розділу потрібно виконувати лише в одному місці моделі; прикладом такого компоненту буде банер веб-сайту або дерево навігації веб-сайту.

Аспект поведінки розглядається в XIS2, WebML та Outsystems шляхом захоплення шаблонів інтерфейсу користувача, що дозволяє програмам взаємодіяти з користувачами, а не лише відображати запитувані ресурси. Крім того, специфікація контролю доступу (тобто, які користувачі або ролі можуть отримати доступ до компонентів інтерфейсу користувача) підтримується у всіх підходах, за винятком UWE та Sketchflow. Хоча в Sketchflow це зрозуміло (з огляду на його призначення, як інструмента створення прототипів), в UWE це перешкоджає моделюванню реальних сценаріїв.

Усі аналізовані підходи дозволяють зв'язувати елементи інтерфейсу користувача та елементи домену (наприклад, налаштовувати рядки таблиці, щоб показати значення поля у конкретних випадках). Проте лише XIS2 та Outsystems дозволяють налаштовувати ці прив'язки в моделі (наприклад, змінити клітинку в рядку таблиці, щоб показати інше значення для кожного

екземпляра). Незважаючи на те, що неможливість налаштовувати ці прив'язки, спрощує модель та дає змогу уникнути помилок, на практиці це є обмеженням мови, що змушує розробників змінювати створений вихідний код для вирішення конкретних вимог.

Підтримка трансформацій моделі в модель - це аспект, який, як правило, не розглядається підходами моделювання веб-додатків; серед аналізованих підходів, лише підходи на основі UML (UWE та XIS2) підтримують даний аспект. UWE та XIS2 використовують трансформації моделей для прискорення задач моделювання шляхом перетворення інформації, яка вже була змодельована в момент здійснення трансформації у нові моделі ("моделі" в UWE та "перегляди" в XIS2). Ці перетворення відображають, як зазвичай моделюються ці представлення, і розробник може змінювати створені моделі відповідно до потреб програми (наприклад, показуючи додаткове поле в інтерфейсі користувача або додаючи додатковий крок у діаграмі активності).

Серед аналізованих підходів лише Outsystems наголошує, що вихідний код не слід редагувати вручну: можна редагувати лише саму модель, а згенерований код і бази даних завжди залишаються недоступними для розробника. Всі інші підходи використовують традиційне програмування (тобто, ручне редагування згенерованих об‘єктів вихідного коду), для того, щоб відповідати певним вимогам, які не можуть бути виражені мовою моделювання підходу.

За винятком Outsystems, всі підходи фактично не залежать від середовища розгортання. Завдяки використанню елементів високого рівня WebML, UWE та XIS2 можуть створювати код для будь-якої веб-орієнтованої платформи (наприклад, ASP.NET, Apache Tomcat); XIS2 також може генерувати код для настільних платформ. Веб-орієнтована версія Sketchflow генерує додаток Silverlight, який, хоча вимагає наявність плагіна Silverlight, встановленого в веб-переглядачі, може розглядатися як крос-платформа,

оскільки вона доступна для платформ Microsoft Windows, Linux і Mac OS X. Нарешті, моделі, створені в OutSystems, можуть бути розгорнуті лише в стеку розгортання OutSystems, які використовують або (1) сервер додатків JBoss і базу даних Oracle, або (2) сервер додатків Microsoft IIS та базу даних SQL Server. Хоча в цьому аспекті платформа OutSystems, очевидно, більш обмежена, ніж інші підходи, це дозволяє автоматизувати більшу частину життєвого циклу розробки веб-додатків, а саме: сценарії розгортання, оновлення і відновлення додатків.

Хоча в даний час більшість веб-додатків все ще розробляються в ручному режимі (наприклад, за допомогою типових завдань програмування), концепція розробки веб-програм за принципом моделювання швидко зростає. Приклади цієї тенденції зростання можна знайти в кількості мов моделювання веб-додатків, у тому числі тих, що були розглянуті в цьому розділі. У цьому розділі було представлено аналіз обраного набору підходів та мов моделювання. Він був зосереджений насамперед на аспектах, які мають відношення до такого роду мов моделювання. З цього аналізу було виявлено низку проблем (які будуть представлені пізніше), а також деякі міркування, які слід враховувати при розробці нових підходів.

2.3. Системи керування вмістом

Незважаючи на те, що ідея керування вмістом існувала з самого початку Інтернету, лише в останні роки ми стали свідками бурхливого розвитку систем CMS. Система керування вмістом (англ. CMS) - це особлива веб-програма, орієнтована на керування та публікацію вмісту (наприклад, публікація блогу, форуму, деякий текст HTML або відео). Ці системи, як правило, надають адміністраторам контенту та споживачам (тобто регулярним користувачам, які просто переглядають веб-сайт) певні можливості, такі як: (1) модульність, (2) незалежність між вмістом та його представленням, (3) управління доступом, (4) користувальницький контроль, або (5) налаштовуваний візуальний вигляд і

компонування вмісту. У цьому розділі аналізуються наступні системи CMS, які є важливими для даної роботи: 1. DotNetNuke1; 2. Drupal2; 3. Joomla3; 4. Vignette Content Management і 5. WebComfort5. Незважаючи на те, що існує постійно зростаюча кількість доступних систем CMS, вищезазначені системи були відібрані для аналізу через значні відмінності між собою. Інші системи CMS (наприклад, WordPress), також придатні для аналізу, але вони дадуть дуже схожі результати - наприклад, аналіз WordPress призводить до майже тих самих значень, що й Joomla -, що робить їх включення в цей аналіз зайвим.

2.3.1. Критерії аналізу

Аналіз, проведений в даному розділі, в основному зосереджений на характеристиках, отриманих з власного досвіду використання CMS систем під час їх дослідження. Більш конкретно, розглянуті критерії охоплюють аспекти, починаючи від адміністративних можливостей (наприклад, конфігурація структури веб-сайту або підтримка «багаторазової оренди») до орієнтованих на розробників функцій, таких як розширюваність або надання конкретного підходу для розробки веб-додатків на базі CMS.

1. Підхід управління. Системи CMS зазвичай використовують один із двох підходів управління: орієнтований на сторінку та орієнтований на вміст. Підхід, орієнтований на сторінку передбачає, що спочатку потрібно визначити структуру веб-сайту (тобто, набір сторінок та компонентів), а потім визначити вміст у контексті цієї структури. З іншого боку, контент-орієнтований підхід передбачає, що вміст повинен бути визначений першочергово, а структура сторінок, які покажуть частину цього вмісту, може бути розроблена адміністратором. Даний аспект аналізує, який з цих підходів управління використовується CMS.
2. Настроювана структура веб-сайту. Цей аспект визначає, чи дозволяє система CMS налаштовувати структуру веб-сайту таким

чином, щоб відвідувачі отримували максимально ефективну організацію веб-сайту у вигляді ієрархічного набору сторінок. Фактично, даний аспект використовується для позначення того, чи підтримує CMS специфікацію структури веб-сайту. В свою чергу, ця структура часто є основою для допомоги відвідувачам при навігації на веб-сайті та доступу до опублікованого контенту.

3. Настроюваний візуальний макет сторінки. Окрім можливості налаштовування структури веб-сайту, адміністратори також повинні мати можливість налаштовувати візуальний макет веб-сайту, який ці сторінки використовуватимуть для показу змісту. Цей аспект визначає, чи підтримує CMS подібний візуальний механізм.
4. Підтримка багаторазової оренди (англ. multitenency). Багаторазова оренда визначає чи можливо створити логічний набір веб-сайтів в межах однієї інсталяції CMS. Іншими словами, цей аспект визначає, чи може окрема інсталяція CMS підтримувати визначення декількох логічних веб-сайтів (наприклад, персональний веб-сайт та веб-сайт електронної комерції), кожен з яких, як правило, доступний за окремою URL-адресою (Universal Resource Locator).
5. Можливість розширення. Цей аспект розглядає механізми, надані системою CMS для його розширення сторонніми особами (наприклад, чи CMS надає API для розробки нових функцій). Цей аналіз зосереджений на наступних питаннях: (1) які мови програмування можуть бути використані; (2) чи можна використовувати функції безпеки, надані CMS, для обмеження можливих дій (або надання додаткової інформації); (3) чи змінюється поведінка за замовчуванням системи CMS; і (4), чи

можна визначати нову поведінку (наприклад, нові компоненти CMS або додатковий код, який запускається під час конкретних подій) у системі.

6. Підхід до розробки. Цей аспект аналізує тип підходу до розробки веб-додатків, який підтримує CMS (навіть якщо веб-додаток складається лише з налаштувань, розширень або чогось, що змінює поведінку системи за замовчуванням). Зокрема, ми визначаємо: (1) чи CMS розглядає будь-який конкретний підхід до розробки веб-додатків; (2) чи цей підхід є модель-орієнтованим.

2.3.2. Результати аналізу

Аналіз цих систем CMS відповідно до перелічених вище критеріїв привів до значень, наведених у Таблиця 2.

Таблиця 2 Характеристики аналізованих CMS систем

	Dot-Net-Nuke	**Drupal**	**Joomla**	**Vignette**	**Web-Comfort**
1. Підхід управління	Орієнт. на сторінку	Орієнт. на вміст	Орієнт. на вміст	Орієнт. на вміст	Орієнт. на сторінку
2. Настроювана структура веб-сайту	+	+	+	+	+
3. Настроюваний візуальний макет сторінки.	+	+	+	+	+
4. Підтримка багаторазової оренди	+	+	-	+	-
5. Можливість розширення	+	+	+	+	+
Мова програмування	C#/ VB.NET	PHP	PHP	Java	C#/ VB.NET
Надає засоби безпеки	+	+	+	+	+
Можна змінити поведінку за замовчуванням	+	-	-	-	+
Можна визначити нову поведінку	+	+	+	+	+
6.Підхід до розробки	Трад.	Трад.	Трад.	Трад.	Трад.

Стосовно підходу управління, який використовується аналізованими системами CMS, можна помітити, що використовуються обидва підходи (орієнтовані на сторінку та орієнтовані на вміст). Більш конкретно, системи DotNetNuke та WebComfort використовують підхід, орієнтований на сторінку, в якому структура веб-сайту (тобто набір вкладок і модулів) визначається першочергово, а потім вміст визначається в модулях. Тим не менше, ці дві системи CMS підтримують обмін вмістом між модулями за допомогою функції копіювання модуля DotNetNuke та функцій копіювання та посилання WebComfort. З іншого боку, системи Drupal та Joomla використовують підхід орієнтований на вміст, в якому адміністратор спочатку визначає вміст, який буде відображатися користувачеві, а потім визначає структуру сторінок, в якій будуть відображатися певні частини доступного контенту. Система Vignette може розглядатися, як сукупність цих двох підходів, оскільки вміст визначається самостійно та представляється користувачеві за допомогою шаблонного механізму Vignette, який використовує заздалегідь визначену структуру веб-сайту для представлення існуючого вмісту.

Всі аналізовані системи CMS дозволяють адміністраторам налаштовувати структуру веб-сайту, як ієрархічний набір сторінок або вузлів (у поєднанні з механізмами зв'язку, наприклад, меню Joomla), залежно від підходу управління, який використовує CMS.

Можливість налаштовувати візуальний макет веб-сайту (зовнішній вигляд веб-сайту, наприклад використовувані кольори або відносне розташування кожного контейнера, яке ці сторінки використовуватимуть для показу вмісту) підтримується всіма аналізованими системами CMS.

Багаторазова оренда (англ. multitenency) підтримується в деяких аналізованих системах CMS. Лише Joomla і WebComfort не підтримують багаторівневу оренду, хоча в обох випадках над цим вже працюють

розробники. Системи CMS, які підтримують цю функцію, зазвичай визначають інший префікс таблиці бази даних для кожного веб-сайту, а це означає, що різні логічні веб-сайти часто є повністю незалежними один від одного.

Всі проаналізовані системи CMS дозволяють розширення сторонніми розробниками. Аспекти, які розробники можуть використовувати або розширювати, різняться, хоча деякі з них є загальними, наприклад, функції безпеки (управління користувачами та ролями). Крім того, усі розглянуті системи підтримують додавання функцій і поведінки, але лише деякі підтримують зміни існуючої поведінки за замовчуванням; в останньому випадку це, як правило, здійснюється за допомогою шаблону розробки стратегії (або його варіанту), а вбудована поведінка за замовчуванням використовується лише, якщо немає іншої стратегії.

Жодна з аналізованих систем CMS не використовує конкретні підходи до налаштування або розробки розширень. Хоча, кожна з цих систем забезпечує деяку підтримку розробників (у різних формах, але зазвичай складається з API для розробки вихідного коду), а сам підхід розробки визначається розробником.

Використання систем CMS у якості базової платформи для створення нових веб-сайтів стрімко зростає. Підтвердженням цього слугує кількость наявних систем CMS та збільшення кількості підтримуваних ними функцій.

У цьому розділі було проаналізовано деякі системи CMS, відповідно до аспектів, які зазвичай є актуальними при розробці веб-застосунків CMS, а також представлено аналіз підходів та мов моделювання, орієнтованих на веб-додатки.

3 БАГАТОМОВНИЙ МОДЕЛЬ-ОРІЄНТОВАНИЙ ПІДХІД ДО РОЗРОБКИ ВЕБ ДОДАТКІВ

Після аналізу сучасних досягнень у області розробки веб-додатків, представленого в попередніх главах, можна зробити висновок, що існує значний розрив між доменом систем CMS, розробкою веб-додатків і використанням модель-орієнтованих підходів до розробки. Цей розрив, у свою чергу, становить серйозну проблему при розробці веб-додатків, які підходять для розгортання в системах CMS, оскільки розробники зазвичай вимушені вибирати між: розробкою веб-додатків на основі CMS за допомогою традиційних способів з усіма проблемами, що властиві таким підходам; або розробкою веб-додатків з використанням модель-орієнтованих підходів до розробки, які часто потребують розробки деяких функцій, які зазвичай надаються у готовому вигляді системами CMS (наприклад, підтримка форумів, блогів або опитувань). У цьому розділі запропоновано підхід, орієнтований на модель-орієнтовану інженерію (англ. MDE), який, ефективно вирішує цю проблему. Поточний розділ розпочинається з короткого аналізу основних проблем, які були виділені на основі аналізу існуючих підходів. Далі, він представляє огляд запропонованого рішення.

3.1 Недоліки існуючих модель-орієнтованих підходів для розробки веб-додатків

Як було зазначено в попередніх розділах, розробка веб-додатків, підтримуваних платформами CMS, зазвичай виконується за допомогою традиційних підходи до розробки програмного забезпечення, в яких вихідний код є основним артефактом, а моделі та документація вважаються допоміжними засобами. Такі підходи забирають багато часу і є схильними до помилок, оскільки вони є залежними від виконання повторюваних задач програмістами. З іншого боку, модель-орієнтовані підходи до розробки

спрямовані на те, щоб виконувати більшість цих повторюваних задач за допомогою автоматизованих трансформацій моделей.

Хоча в даний час існують деякі модель-орієнтовані підходи для розробки веб-додатків, підходів для розробки веб-застосунків на базі CMS немає. Ідея використання систем CMS як платформ для більш складних веб-додатків є відносно недавньою, але дані системи забезпечують певну ступінь розширюваності та ефективності розробки, яку варто використати.

Більш того, більшість модель-орієнтованих підходів до розробки веб-додатків не забезпечують належну підтримку різних точок зору, що мають зацікавлені сторони стосовно розроблюваного додатку. Ці точки зору часто:

1. фіксуються вручну (використовуючи засоби, такі як документи з вимогами та зустрічі із зацікавленими сторонами);
2. вручну інтерпретуються командою розробників (включаючи системних архітекторів та експертів доменів);
3. надаються програмістам для розробки конкретної системи.

Як правило, вважається, що вирішення різних точок зору легше робити за допомогою природної мови, що є мовою, на якій люди все ще набагато вправніші, ніж комп'ютери.

Іншою проблемою в сучасних підходах MDE є те, що вони:

1. включають в мову моделювання занадто багато деталей низького рівня, намагаючись зробити її більш виразною;
2. намагаються включити якомога менше даних про низький рівень в мову, щоб полегшити її вивчення та використання.

Хоча це здається очевидним (адже було б дуже важко, якщо не неможливо, визначити мову моделювання, достатньо виразну для потреб кожного учасника процесу створення системи), це фактично показує необхідність компромісу, який повинен бути прийнятий через проблему, згадану в попередньому пункті.

Нарешті, кінцевою метою більшості сучасних підходів MDE є отримання вихідного коду замість моделей. Це часто призводить до того, що розробники змінюють сгенерований вихідний код, тоді як сама модель залишається незмінною (і, таким чином, не синхронізованою з вихідним кодом). Незважаючи на те, що існують методи, які дозволяють ручні зміни вихідного коду при збереженні синхронізації з моделлю, розробники досі, як правило, мають необхідність вручну редагувати вихідний код, якщо мова моделювання не є достатньо виразною.

3.2 Модифікація існуючого модель-орієнтованого підходу для розробки веб-додатків: багатомовний підхід

Для вирішення виявлених проблем ми пропонуємо підхід, орієнтований на MDE, для розробки веб-додатків на основі CMS. Цей підхід передбачає деякі помітні відмінності від інших підходів, орієнтованих на MDE для розробки веб-додатків, а саме: (1) він базується на використанні декількох мов моделювання; і (2) він використовує механізм синхронізації моделей, щоб: а) забезпечити узгодженість між моделями різних мов; б) дозволити зацікавленим сторонам одночасно змінювати різні типи моделей без необхідності турбуватися про можливу втрату інформації. На рис. 10 наведено спрощений огляд запропонованого підходу.

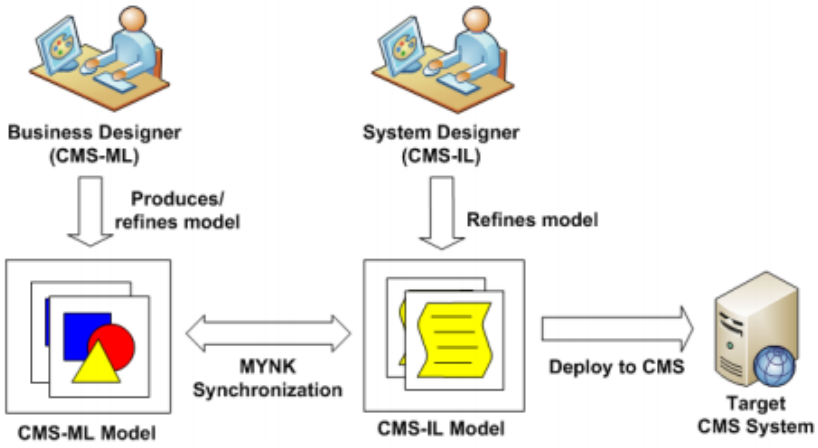

Рисунок - 10 Огляд запропонованого методу розробки, орієнтованого на MDE

Замість визначення однієї мови моделювання, орієнтованої на CMS системи, цей підхід визначає дві мови: (1) CMS-IL (CMS Intermediate Language) - мова низького рівня, що забезпечує спільну основу для специфікації веб-додатків на основі CMS; і (2) CMS-ML (мова моделювання CMS), яка надає набір елементів, які можуть бути використані для визначення моделі високого рівня веб-додатка. Крім мов CMS-ML та CMS-IL, підхід також розглядає мову MYNK, яка підтримує механізм синхронізації моделі.

Слід зазначити заздалегідь, що мови CMS-IL та CMS-ML не є достатніми для підтримки всіх учасників процесу розробки. Натомість, аби підтвердити твердження даної роботи, ми зосереджуємося на досить обмежених типах учасників процесу, які є достатньо репрезентативними для визначення людей, які зазвичай беруть участь у проектах розробки програмного забезпечення. Через дані обмеження типів учасників, ми також пропонуємо короткий набір інструкцій для створення нових мов моделювання, які можуть бути використані з механізмом синхронізації моделі. Ці рекомендації можуть бути використані при визначенні нових мов моделювання відповідно до потреб додаткових видів зацікавлених сторін.

3.2.1 Мова моделювання CMS-ML

CMS-ML - це графічна мова моделювання, яка надає засоби для специфікації веб-застосунків на основі CMS на високому рівні та незалежно від платформи. Мета даної мови - дозволити нетехнічним зацікавленим особам швидко моделювати веб-додаток, підтримуваний CMS системами, а саме за рахунок надання засобів для легкого читання, розуміння та змінення моделі. Крім того, CMS-ML також повинна мати механізм розширюваності, спрямований на підтримку спеціальних вимог зацікавлених сторін та моделювання веб-додатків з більш високим ступенем складності. Цільовою аудиторією для CMS-ML, як вже зазначалося, є нетехнічні учасники процесу, які моделюватимуть, яким саме повиннен бути веб-додаток.

3.2.2 Мова моделювання CMS-IL

CMS-IL - це текстова мова низького рівня, що має вищий рівень виразності, ніж CMS-ML, та є незалежною від конкретної платформи CMS. Ця мова забезпечує елементи структурного моделювання, але, на відміну від CMS-ML, також зосереджується на тому, як веб-додаток повинен працювати. Таким чином, CMS-IL визначає набір елементів моделювання, які дозволяють деталізувати веб-додаток на рівні абстракції близькому до реалізації.

Метою CMS-IL є створення спільної основи для специфікації веб-додатків на основі CMS, на відміну від CMS-ML, метою якої є забезпечення простого способу створення моделей веб-додатків. Цільовою аудиторією для CMS-IL є технічні учасники процесу (а саме, розробники). Звичайно, ці розробники повинні бути знайомі з концепціями, які зазвичай можна знайти в системах CMS.

3.2.3 Процес розробки

Робочий процес, який вже було показано на рис. 8, розглядає два основних типи зацікавлених сторін, які ми позначаємо як бізнес-конструктор та системний конструктор.

Бізнес-конструктор - загальний термін для ідентифікації нетехнічних учасників - може почати робочий процес, створивши модель CMS-ML, яка представляє цільовий веб-додаток, відповідно до деяких заздалегідь визначених бізнес-вимог (рис. 11).

Після використання MYNK для отримання моделі CMS-IL (з вищезгаданої моделі CMS-ML), системний конструктор - який, на відміну від бізнес-конструктора, використовується для ідентифікації технічних учасників, повинен визначити, чи ця модель CMS-IL є задовільною шляхом виявлення будь-яких особливих вимог, які не можуть бути вирішені лише за допомогою CMS-ML. Якщо такі вимоги існують, отримана модель CMS-IL повинна бути додатково модифікована системним конструктором для їх вирішення. У той

час, як конструктор системи змінює модель CMS-IL, механізм синхронізації MYNK повинен підтримувати моделі CMS-IL та CMS-ML (які відповідають одному і тому ж веб-застосуванню, розглянутому в перспективі різних зацікавлених сторін) узгодженими між собою.

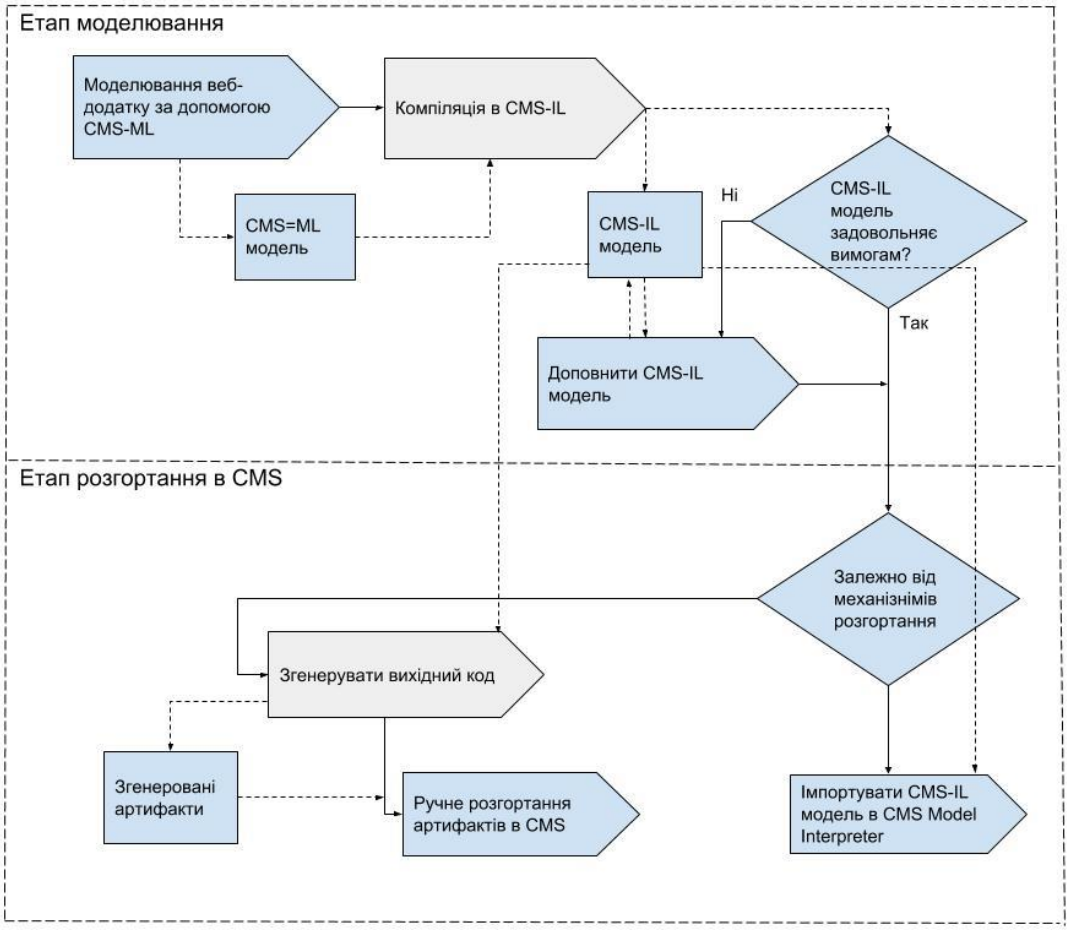

Рисунок - 11 Запропонований модель-орієнтований підхід

Після процесу уточнення модель CMS-IL повинна бути точним представленням того, яким повинен бути веб-додаток. Проте, якщо бізнес-конструктор не погоджується з моделлю CMS-ML (яка була автоматично оновлена через MYNK, щоб відобразити зміни, зроблені системним конструктором у відповідній моделі CMS-IL), тоді доведеться змінити модель CMS-ML, запустивши ще одну ітерацію механізму синхронізації MYNK.

Коли модель CMS-IL (і, відповідно, модель CMS-ML) вважається задовільною, вона розгортається на цільовій системі CMS одним із двох

способів, залежно від самої CMS: (1) розгортання за допомогою компонента CMS Model Interpreter, який вже доступний у системі CMS, або (2) генерація об‘єктів низького рівня та подальша установка.

Компонент CMS Model Interpreter є компонентом, який встановлений у системі CMS, і відповідає за (1) отримання вхідної CMS-IL моделі та (2) розгортання отриманої моделі шляхом конфігурації системи CMS для відображення структури та об‘єктів, що визначених в моделі.

Хоча даний підхід не усуває потребу в ітеративному процесі розробки, ми вважаємо, що це потенційно зменшує кількість додаткової роботи, пов'язаної з розбіжністю між перспективами учасників процесу створення системи та розумінням цих перспектив розробником.

Слід зазначити, що ми не виключаємо можливий сценарій, в якому (1) бізнес-дизайнер визначає модель CMS-ML, а потім (2) розгортає його на компоненті CMS Model Interpreter цільової системи CMS, минаючи уточнення моделі конструктором системи (наприклад, для швидкого отримання прототипу потрібної веб-програми).

3.2.4 Рекомендації щодо специфікації мови

Як вже згадувалося, підхід, запропонований в цій роботі, передбачає лише дві мови, CMS-ML та CMS-IL. Однак буде невірно припустити, що цих двох мов буде достатньо для підтримки будь-якого зацікавленого учасника розробки веб-додатка. Це, в свою чергу, підвищує ймовірність виникнення випадків, коли підхід має бути розширений за допомогою додаткових мов моделювання, розроблених для підтримки нових точок зору зацікавлених сторін. Тому виникає необхідність визначити правила створення таких мов.

У ході досліджень, представлених у цій роботі, було визначено невелику кількість рекомендацій. Хоча ці рекомендації не слід тлумачити, як кроки в методології визначення мов, ми вважаємо, що вони можуть допомогти у вирішенні деяких потенційних проблем, які зазвичай виникають під час

процесу розробки мови, наприклад, проблеми метамоделювання або декількох альтернатив конкретного синтаксису. Крім того, слід зазначити, що деякі з цих правил були засновані на правилах визначення мови DSM, що містяться в , тоді як інші були засновані на найкращих практиках дизайну DSL .

Визначені принципи пояснюються нижче (передбачається, звичайно, що розробник мови вже шукав DSL, який вирішує проблеми бажаного домену, і не знайшов відповідних мов).

Визначити цільову аудиторії для мови. Це, на мій погляд, одна з найважливіших рекомендацій, оскільки її результати вплинуть на більшость майбутніх рішень щодо розробки мови. Зокрема, важливо з'ясувати:

1) Основні проблеми аудиторії щодо цілей моделювання мови;

2) Очікуваний рівень технічної експертизи аудиторії (враховуючи, що мета використання мови полягає в отриманні моделі, яка точно відображає програмне забезпечення).

Хоча вищенаведена рекомендація, мабуть, є очевидною, тим не менш, це важливо, тому що кінцева мета моделі полягає в тому, щоб відповідати на питання про систему, яку вона представляє . Таким чином, важливо, перш за все, визначити, на які питання необхідно відповісти, і лише після цього визначити спосіб, як відповідати на ці питання.

Визначити ступінь розширюваності, який повинна мати мова. Після визначення проблемної області розробник мови повинен враховувати, чи потрібно буде змінювати мову (наприклад, додавши нові елементи або змінюючи елементи, які вже присутні в мові).

У , Стендіш виділяє три різні підходи до розширюваності мови програмування: (1) перефразування, в якому нові функції додаються до мови, за допомогою функцій, які вже присутні на цій мові; (2) ортофразування, в якому нові функції додаються до мови, не вимагаючи функцій вже включених до мови; і (3) метафразування, в якому змінюються правила інтерпретації мови,

що приводить до іншої обробки існуючих мовних функцій. Хоча ці підходи були виявлені в контексті розробки мов програмування, можна вважати, що такі мови програмування є текстовими випадками мов моделювання. Таким чином, ці підходи також можуть бути застосовані до розширюваності дизайну мови моделювання.

Враховуючи визначену проблемну область, визначити рівні моделювання мови та їх ієрархію. Незважаючи на те, що дана рекомендація тісно пов'язана з попереднім принципом відносно розширення, основна увага зосереджена на дещо іншому. Попередня рекомендація спрямована, перш за все, на визначення того, які аспекти мови повинні бути розширювані своїми користувачами, тоді як поточним керівним принципом є структуризація концептів мови (з проблемного домену) в різні рівні моделювання (тобто метарівні).

Більш конкретно, розробник мови повинен розглянути різні концепти та відповідні об'єкти (раніше виявлені під час аналізу проблемної області), також визначити (1), що він класифікує, і (2) що є будівним блоком в даному випадку.

Це випливає з поняття стратифікованого проектного підходу . У такому підході складна система повинна бути структурована відповідно до ряду рівнів, описаних з використанням різних мов. Кожен рівень визначається шляхом об'єднання елементів, які вважаються примітивними на тому рівні, і такі примітиви самі є результатом комбінацій, виконаних на попередньому рівні. Автори також заявляють, що мова на кожному рівні має містити (1) примітиви, а також (2) комбінацію та (3) абстрактні механізми, відповідні до необхідного рівня деталізації.

Чітке визначення цих рівнів моделювання (метарівнів в номенклатурі метамоделювання) дозволяє конструктору мови встановити ієрархічну структуру між цими метарівнями. Ця ієрархія, у свою чергу, забезпечує практику строгого метамоделювання , якщо конструктор мови забезпечує те,

що: (1) у кожному метарівні не існує жодного індексу зв'язку; (2) екземпляр є єдиним зв'язком між елементами різних метарівнів; і (3) всі елементи EL1, що виникають у метарівні ML1, пов'язані (через екземпляр зв'язку) з елементами EL2 в іншому метарівні ML2.

Визначити будь-які обмеження, які можуть зумовлювати вибір мови метамоделювання. Крім концепцій та взаємозв'язків проблемного домену, розробник мови повинен також визначити будь-які обмеження, які можуть зробити певні мови метамодування непридатними для визначення абстрактного синтаксису мови.

Хоча більшість мов метамоделювання не накладають значного набору обмежень через їх простоту, розробники мови повинні все-таки визначити відповідні обмеження (пов'язані з заданою областю проблем або визначеними рівнями моделювання), а також оцінити, чи здатна вибрана мова метамодування моделювати або забезпечити ці обмеження задовільним чином. Можливими прикладами таких обмежень є використання множинного наслідування (наприклад, мова метамоделювання Kermeta1 підтримує множинне успадкування, включаючи механізм для вирішення конфліктів стосовно оператора та перевизначення функції), або необхідність вказання конкретних обмежень для конкретної моделі (наприклад, щоб моделювати той факт, що автомобіль повинен мати чотири колеса).

Окрім виявлених обмежень, розробники мови також повинні оцінювати переваги та недоліки відповідних проблем при розгляді питання про використання (1) універсального моделювання, такого, як UML, або (2) мови, яка орієнтована на DSM, для метаметамоделі мови. Кожен з цих підходів матиме свої переваги та недоліки. Одним з важливих питань є підтримка інструментів, оскільки прийняття універсальної мови замість DSM може посприяти створенню інструмента моделювання. Іншим питанням є підтримка моделей мови (тобто можливість нових користувачів правильно розуміти та

змінювати модель), щодо якого показує перевагу при використанні DSM замість UML.

Визначити тип конкретного синтаксису, який буде найбільш комфортним для цільової аудиторії. Ця рекомендація спрямована на вирішення питання щодо того, чи має мова мати графічний, текстовий синтаксис, використовувати змішаний підхід до синтаксису мови (шляхом надання як графічних, так і текстових елементів), або навіть шляхом надання декількох синтаксисів. Це, у свою чергу, має вирішальне значення для забезпечення якості та зручності використання мови і у підсумку її прийняття користувачами
.

Для цього розробник мови повинен враховувати цільову аудиторію та її бажану форму представлення потрібної системи. Технічна експертиза аудиторії також може бути фактором, що впливає на це рішення: відповідно до (1) значна частина розробників віддає перевагу текстовим мовам через їхню діяльність, пов'язану з традиційним програмуванням та скриптовими мовами, тоді як (2) нетехнічні користувачі, як правило, віддають перевагу графічним мовам, якщо вони відображають форми реальних сутностей, які представлені в моделі. Потрібно також враховувати частоту, з якою буде використовуватися мова. Мова, яка часто використовується, має надавати синтаксис, який гарантує, що його користувачі зможуть визначати моделі швидко та без помилок, але це, швидше за все, не є проблематичним для мов, які використовується лише іноді.

У цьому розділі представлено модель-орієнтований підхід до розробки веб-застосунків на основі CMS. Цей підхід відрізняється від інших існуючих підходів до розробки веб-додатків шляхом (1) використання двох мов моделювання (замість однієї мови, наприклад, UML або WebML), і (2) механізму синхронізації. Більш конкретно, підхід визначає мови CMS-ML, CMS-IL та MYNK. CMS-ML та CMS-IL, призначені для використання бізнес-

конструкторами (зацікавленими сторонами бізнесу) та системними конструкторами (програмістами), відповідно. Мова синхронізації моделі MYNK призначена для використання розробниками мов (наприклад, розробниками DSL, які хочуть визначити нові мови, орієнтовані на CMS, відмінні від CMS-ML та CMS-IL), як спосіб дозволити іншим видам зацікавлених сторін брати участь у процесі розробки веб-додатків.

Також було представлено невеликий набір інструкцій для визначення нових мов моделювання, які можуть бути використані з MYNK. Ці рекомендації дозволяють створювати додаткові мови, які краще задовольняють потребам інших зацікавлених сторін, що може виявитися корисним у тих випадках, коли CMS-ML та CMS-IL не є достатніми для створення веб-додатку.

4 ВКАЗІВКИ ЩОДО СТВОРЕННЯ МОВ МОДЕЛЮВАННЯ ТА ЗАСОБУ СИНХРОНІЗАЦІЇ МОДЕЛЕЙ

4.1 Мова моделювання CMS-ML

Типовою проблемою сучасних мов моделювання веб-додатків є те, що їх моделі часто є обчислювально-орієнтованими (тобто орієнтовані на специфічні деталі програмування) і не є зрозумілими для нетехнічних користувачів в тому сенсі, що такі люди не зможуть легко визначити, яку інформацію передає модель. Ця проблема виникає в більшості підходів моделювання веб-додатків (таких як UWE , XIS2 або WebML), деякі з яких вже були проаналізовані в розділі 2.

Щоб вирішити цю проблему, ми пропонуємо створити графічну мову моделювання CMS-ML, яка б дозволяла визначати веб-додатки на основі CMS на високому рівні та незалежно від платформи. Крім того, ця мова також повинна дозволяти її розширення, з метою вирішення потреб зацікавлених сторін та підтримки моделювання веб-додатків з більш високим ступенем складності.

Мова CMS-ML повинна дозволити нетехнічним користувачам швидко моделювати веб-додаток, підтримуваний системою CMS. Головною задачею даної мови є дозволити користувачам:

1. Легко сприймати модель у тому сенсі, що модель не повинна бути настільки складною, щоб користувач міг легко загубитись, дивлячись на неї;
2. Розуміти модель, тобто правильно інтерпретувати семантику моделі;
3. Змінювати модель, щоб вона більш точно відображала наміри користувача.

4.1.1 Рекомендації

Перш, ніж почати опис мови CMS-ML, важливо спочатку встановити відповіді на питання, зазначені у розділі 3. У наступних пунктах наведені рекомендації щодо керівних принципів визначення мови моделювання.

Цільова аудиторія для CMS-ML – це бізнес-користувачі, з різним ступенем технічного знання. Ці користувачі усвідомлюють (з огляду на їх повсякденну діяльність у роботі з веб-браузерами та веб-додатками на базі CMS): основні концепції CMS, такі як Веб-сайт, Сторінка, Роль та Компонент (Віджети); основні HTML елементи вищезазначених компонентів, такі, як Посилання, Кнопка, Поле для вводу та Список; і специфікації діяльностей, що моделюються у вигляді блок-схем чи аналогічних діаграм. Проте вони не знайомі з типовими мовами та діяльностями програмування, що часто призводить до семантичного розриву між ними та розробниками .

Відповідно до проблемної області, визначеної в попередньому пункті, зацікавленим особам потрібно мати можливість визначати нові види компонентів CMS (які будуть згодом інстальовані на модельованому веб-сайті). Однак користувачам не доведеться змінювати компоненти, які вже присутні в CMS. Це пояснюється тим, що: (1) більшість систем CMS не дозволяє додаткові налаштування компонентів, що входять в систему; і (2) такі компоненти, як правило, створюються з метою вирішення загальних проблем, а не конкретних проблем, які має цільова аудиторія. Ці проблеми, у свою чергу, часто вимагають створення абсолютно нових компонентів.

Враховуючи відповіді, отримані в попередніх пунктах, необхідно визначити ввідношення композиції: (1) між Компонентом та його компонентами HTML (при визначенні нових Компонентів); (2) між робочим процесом та його діями; і (3) між веб-сайтом, його сторінками та їх компонентами.

Цільова аудиторія ML не складається з розробників (які зазвичай використовують для роботи текстові мови програмування), і тому графічна мова моделювання може бути кращим вибором . Проте, не обов'язково, щоб ця мова була суто графічною, вона може містити деякі фрагменти тексту. Окрім того, цільова аудиторія може бути дещо знайома з графічними мовами моделювання, а саме UML та діаграмами діяльності.

4.1.2 Типи моделей та ролі моделювання

Для моделювання веб-додатків за допомогою CMS-ML пропонується використовуваи три типи моделей: (1) шаблони веб-сайтів (англ. WebSite Templates), (2) анотації веб-сайтів (англ. WebSite Annotations) та (3) набори інструментів (англ. Toolkits). На рис. 12 зображено як ці моделі взаємопов'язані: шаблони веб-сайтів та набори інструментів можуть посилатися на інші набори інструментів, але анотації веб-сайтів можуть лише декорувати шаблони веб-сайтів.

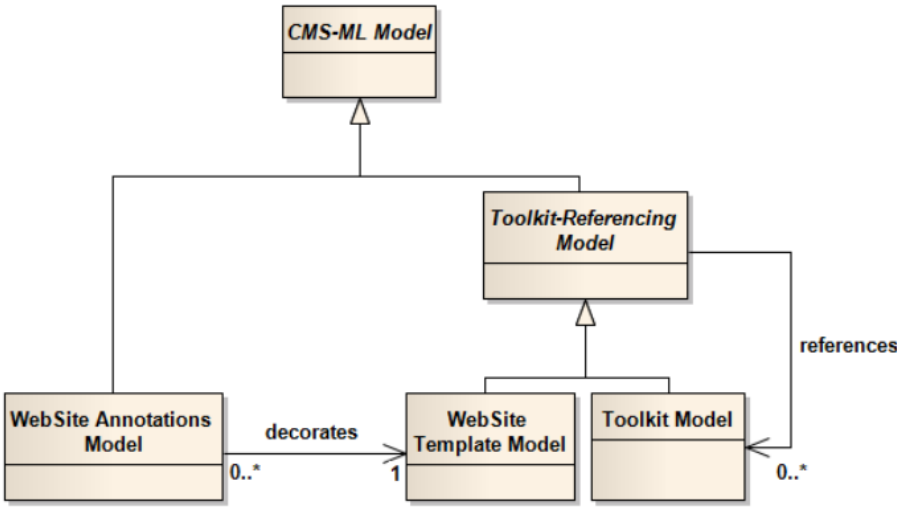

Рисунок - 12 Відношення між різними моделями в CMS-ML

Шаблон веб-сайту - це модель, яка відображає необхідну структуру веб-додатка; цей шаблон моделюється за допомогою елементів CMS, таких як Role, Dynamic WebPage та WebComponent, які надаються CMS-ML.

Набір інструментів дозволяє визначити нові елементи моделювання орієнтовані на CMS, шляхом визначення моделі домену, інтерфейсу користувача та відповідної поведінки. Як згадувалося раніше, шаблон веб-сайту може посилатися на набір інструментів. Крім того, набір інструментів може також посилатись на інші набори інструментів, надаючи можливість сценаріїв, в яких один набір покращує та розширює функціональність, раніше визначену в іншому наборі.

Елементи шаблону веб-сайту можуть бути доповнені за допомогою анотацій. Ця модель дозволяє розробникам шаблонів визначати властивості CMS (наприклад, конфігураційні параметри), не переповнюючи сам шаблон специфічними для платформи деталями.

Не обов'язково, щоб один CMS-ML розробник мав навички для створення моделей шаблонів та наборів інструментів. Натомість ми вважаємо, що розробка за допомогою CMS-ML часто буде виконуватися відповідно до ролей, показаних на Рис. 13:

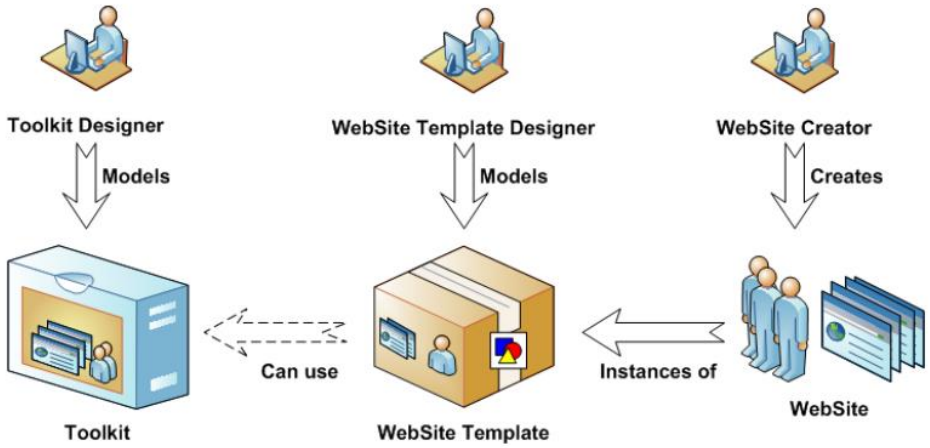

Рисунок - 13 Ролі моделювання в CMS-ML

1. Розробник набору інструментів, який моделює набір інструментів;
2. Розробник шаблону веб-сайту, який моделює шаблон веб-сайту, а також, якщо це можливо, доповнює його за допомогою моделі анотацій;

3. Створювач веб-сайту, який створює екземпляри різних елементів, визначених у шаблоні веб-сайту.

4.1.3 Архітектура

Розглядаючи архітектуру CMS-ML важливо зазначити, що шаблони веб-сайтів та набори інструментів розташовані на різних метарівнях. Шаблони призначені для створення абстракцій (тобто моделей) конкретних веб-додатків, використовуючи елементи, орієнтовані на CMS, а набори інструментів використовують загальні елементи моделювання для створення нових елементів моделювання, орієнтованих на CMS. Рис. 14 зображує метарівні, що визначають CMS-ML.

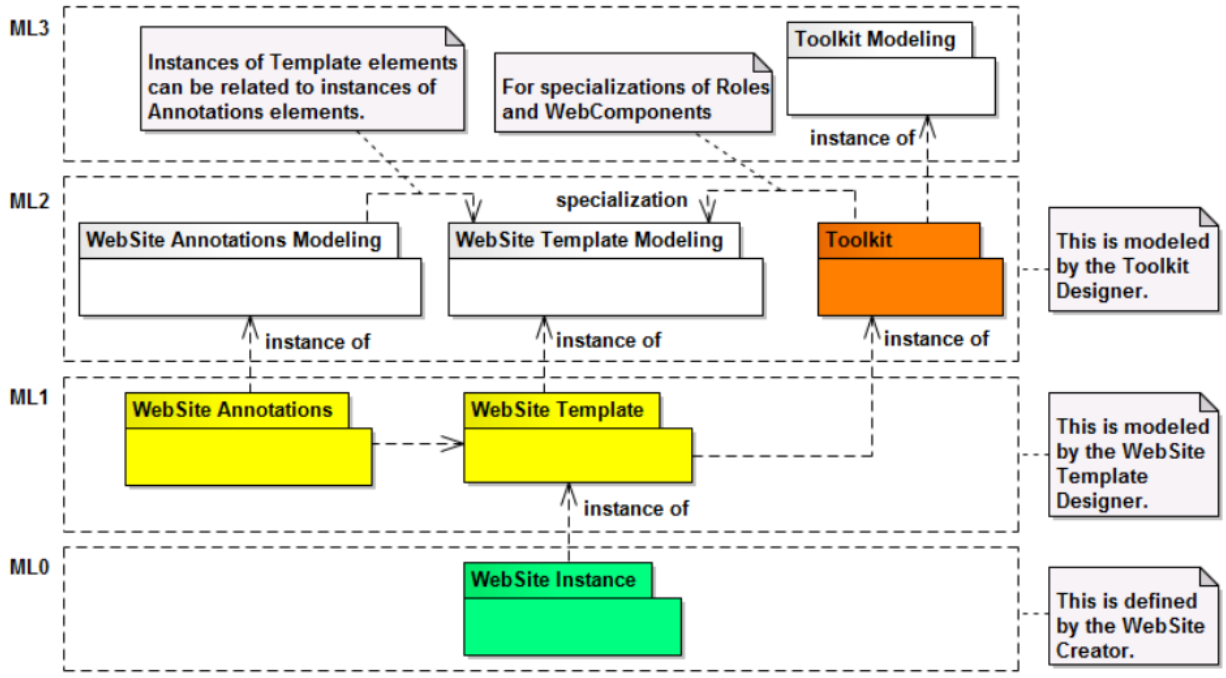

Рисунок - 14 Метарівні CMS-IL

1. Метарівень ML3 містить модель моделювання інструментарію, яка надає визначення загальних елементів моделювання, які будуть використовуватися для визначення моделей наборів інструментів. Цей метарівень є незмінним;

2. Метарівень ML2 містить моделі моделювання веб-сайтів та анотацій, які надають елементи моделювання, які будуть використовуватися для визначення моделей шаблонів веб-сайтів та анотацій WebSite. Подібно до моделей моделювання інструментарію в ML3, моделі моделювання шаблонів анотацій фіксуються і є незмінними;
3. Метарівень ML1 дозволяє створити моделі шаблонів та анотацій, використовуючи елементи моделювання, визначені в ML2.
4. На метарівні ML0 створювач веб-сайту використовує елементи, визначені в моделях шаблону та анотацій веб-сайту для налаштування певного встановлення у CMS. Як правило, це вимагає певного механізму CMS, який встановлює взаємозв'язок між моделлю та екземпляром.

Слід зазначити, що існує деяка схожість між метарівнями ML1 та ML0 та метарівнем M1, що міститься в OMG специфікації UML, оскільки їх призначення практично однакове. Основна відмінність полягає в тому, що, хоча в UML-екземплярах класи та об'єкти розташовані в одному метарівні (M1), в CMS-ML вони знаходяться в метарівнях ML1 та ML0, відповідно.

Основними причинами для даної метарівневої архітектури були: (1) надання засобів для розширення мови в простій формі; (2) зменшення випадкової складністі , яка зазвичай виникає через використання патернів моделювання подібних до тип-екземпляр на одному метарівні; і (3) дотримання строгої доктрини метамоделювання , в якій не повинно бути прикладу відносин типу «екземпляр», що перетинають більше одного метарівня.

4.1.4 Опис моделі шаблону веб-сайта CMS-ML

Мова CMS-ML надає набір елементів орієнтованих на CMS, які розробники можуть використовувати для визначення шаблонів веб-сайтів на

базі CMS. Модель «шаблон веб-сайту», складається з екземплярів цих елементів CMS та визначається відповідно до наступної стуктури (рис. 15):

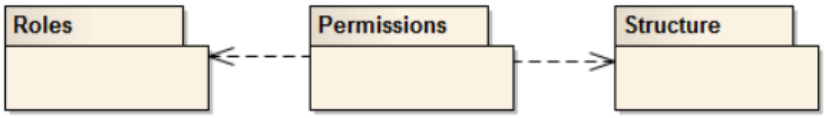

Рисунок - 15 Структура шаблону веб-сайту

1. Структура - визначає структурні компоненти веб-додатка;
2. Ролі – визначає сукупність обов'язків, які веб-додаток очікує від користувачів;
3. Дозволи - визначає, які ролі мають доступ до структурних компонентів веб-додатка.

«Структура» є найбільш важливим елементом моделі шаблону веб-сайту, оскільки вона відразу визначає структуру сторінки веб-додатка, використовуючи набір концепцій, орієнтованих на CMS: (1) Веб-сайт, який представляє екземпляр веб-додатка і виконує роль контейнера для динамічних веб-сторінок; (2) Динамічна веб-сторінка, що представляє динамічно згенеровані сторінки (в тому сенсі, що їх зміст можна змінити через інтерфейс CMS); (3) Контейнер, який моделюється в межах певної області динамічної веб-сторінки та містить набір Веб-компонентів; і (4) Веб-компонент, що представляє частини функціональності (наприклад, блог або форум), з якими користувач буде взаємодіяти. На рис. 16 та рис. 17 представлено абстрактний та конкретний синтаксис «структури» шаблону веб-сайта.

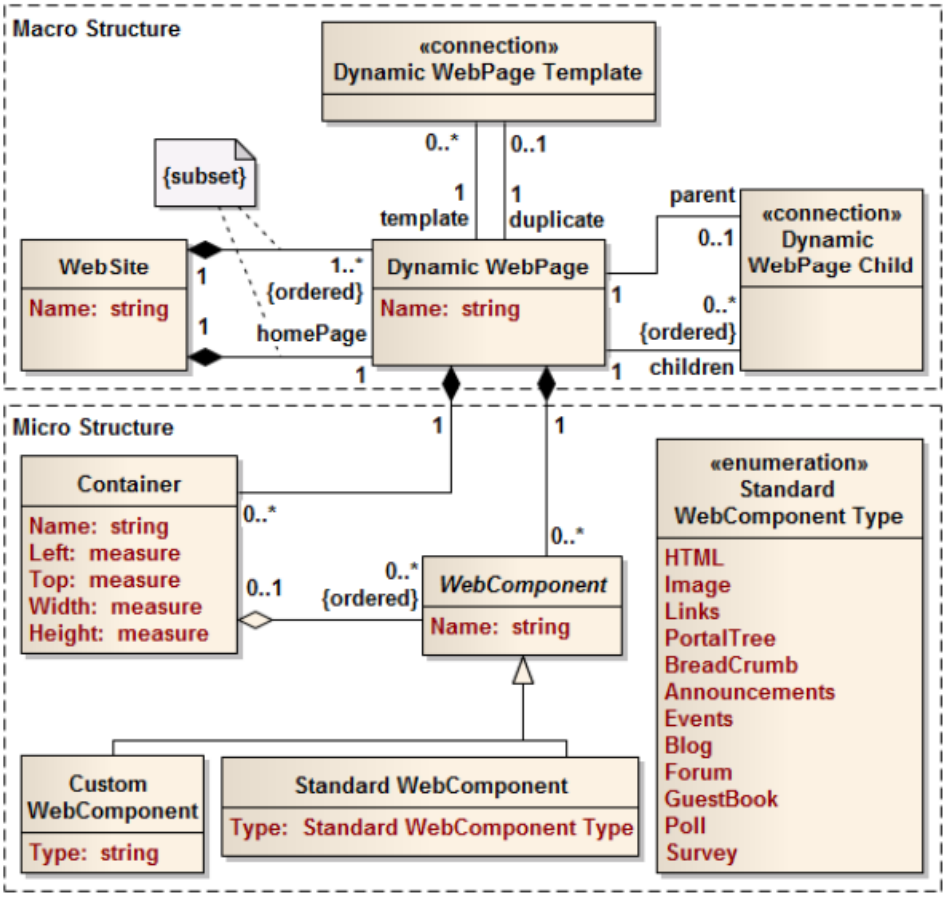

Рисунок - 16 Абстрактний синтаксис "структури" шаблону веб-сайта

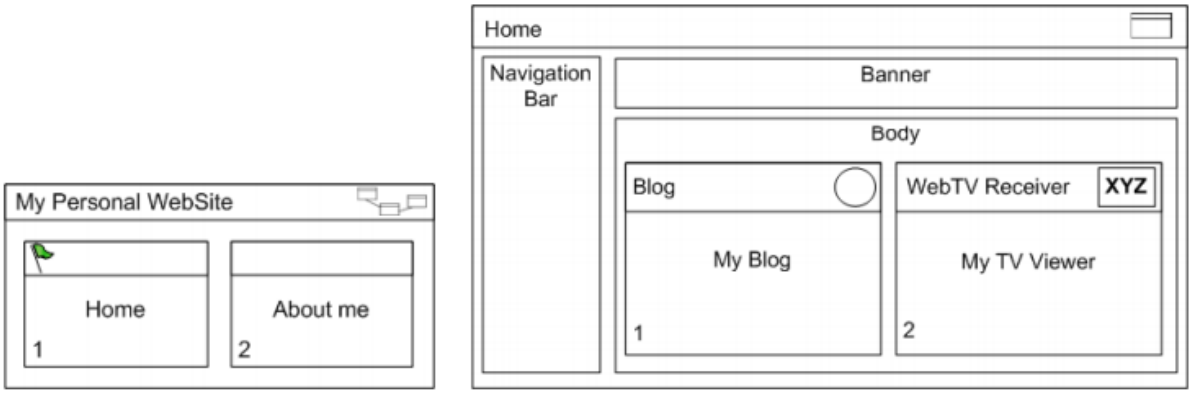

Рисунок - 17 Конкретний синтаксис "структури" шаблону веб-сайта

Представлення «ролей» описує різні види відповідальності, які будуть мати користувачі веб-сайта на основі CMS. Це представлення визначає поняття ролей та делегації ролей. Рис. 18 ілюструє абстрактний синтаксис для

представлення ролей. На рис. 19 наведено конкретний синтаксис для кожного з цих понять.

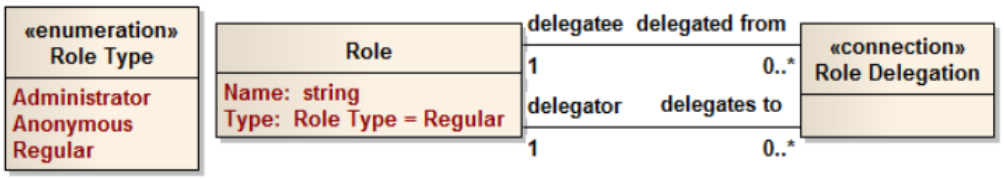

Рисунок - 18 Абстрактний синтаксис представлення ролей для шаблону веб-сайта

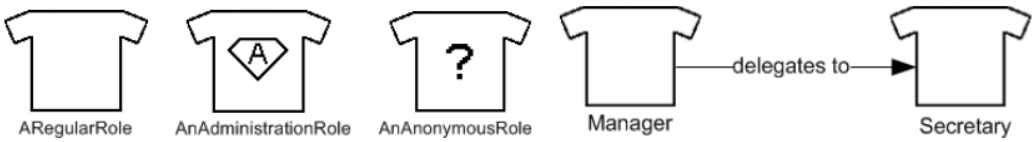

Рисунок - 19 Конкретний синтаксис представлення ролей для шаблону веб-сайта

Представлення дозволів встановлює зв'язок між структурою веб-додатка та його ролями, а саме, визначає, які дії можуть виконувати користувачі. Визначається два поняття: Динамічний дозвіл веб-сторінки і Дозвіл веб-компонента, які, відповідно, дозволяють створювати зв’язки роль-динамічна веб-сторінка та роль-веб компонент. На рис. 20 проілюстровано ці поняття.

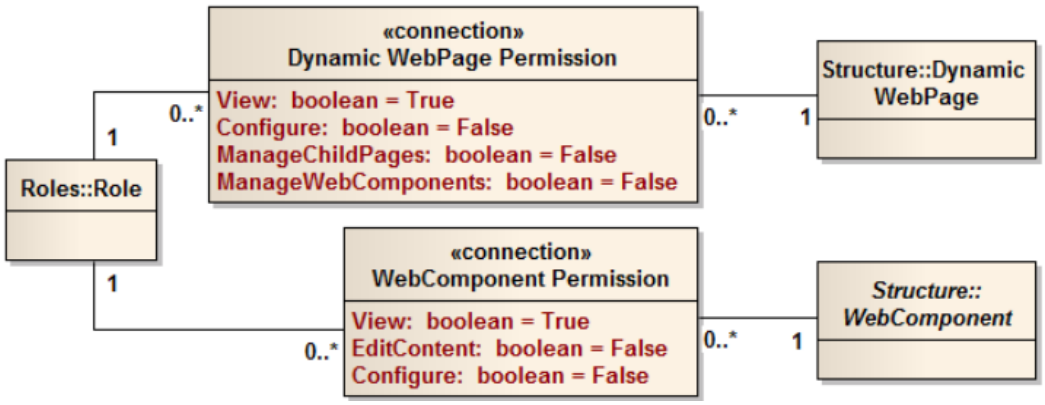

Рисунок - 20 Абстракний синтаксис представлення ролей

4.2 CMS-IL

Мова CMS-ML дозволяє нетехнічним користувачам моделювати веб-додаток на основі CMS відповідно до їх намірів, на високому рівні та незалежно від платформи. Проте мова є компромісом між можливістю легкого вивчення та кількістю передбачених елементів моделювання. Основною передумовою цього рішення є основна мета мови моделювання - дозволити нетехнічним користувачам (тобто людям, які не мають досвіду розробки та підтримки веб-додатків) швидко розробити правильні моделі веб-додатків на базі CMS. Тим не менш, це компроміс робить CMS-ML нездатною вирішувати особливі вимоги, які, як правило, очікуються від веб-додатків, наприклад специфікацію алгоритмів або інтеграцію з зовнішнішніми веб-сервісами.

Щоб вирішити цю проблему, ми пропонуємо CMS-IL (CMS Intermediate Language) - текстову мову, яка забезпечує низький рівень абстракції над обчислювальними поняттями (в тому сенсі, що вона схожа на мову програмування). Метою цієї мови є:

- Забезпечити механізм, незалежний від будь-якої конкретної реалізації CMS, який може бути використаний технічними користувачами для (1) вирішення проблем низького рівня, які не можуть бути оброблені CMS-ML, і (2) розгортання моделі веб-додатка на будь-якій платформі CMS (якщо платформа може інтерпретувати моделі CMS-IL);
- Створити спільну основу для специфікації веб-додатків на базі CMS.

4.2.1 Рекомендації

У наступних пунктах наведені рекомендації щодо керівних принципів визначення мови моделювання, визначених у розділі 3.

Цільова аудиторія для CMS-IL - це технічні користувачі (а саме розробники), які добре освічені в галузі веб-додатків на базі CMS. Ці користувачі знають загальні CMS концепції, які використовуються бізнес-

користувачами CMS-ML. Бізнес-користувачі визначають новий компонент CMS, вказавши, яку проблему повинен вирішувати компонент, а користувачі CMS-IL визначають низький рівень того, як компонент досягне своєї мети.

Як і в випадку з CMS-ML, проблемним доменом для цієї мови є системи CMS і веб-додатки на базі CMS. Однак, через свою цільову аудиторію, CMS-IL орієнтована на специфікацію того, як вирішувати задачі, які були змодельовані в CMS-ML; у свою чергу, це вимагатиме, щоб CMS-IL надавала технічні концепції (а саме, представляла інструкції для керування комп'ютером), яких немає в CMS-ML. Інші аспекти, які слід враховувати, включають специфікацію користувачів веб-додатку, вмісту і зовнішнього вигляду, а також локалізацію та структурування вищезазначеного вмісту.

Як і в CMS-ML, так і відповідно до виявленої проблемної області, користувачі CMS-IL повинні мати можливість додавати нові види компонентів CMS, які будуть підтримувати задачі, які раніше були ідентифіковані в CMS-ML. Користувачам CMS-IL потрібно буде додавати фрагменти вихідного коду (написаного на мові програмування специфічної для конкретної CMS), а саме, виклики та допоміжні функції, які взаємодіють безпосередньо з основною системою.

Крім того, користувачі повинні також мати можливість визначати вихідний код (за допомогою CMS-IL іншої CMS-специфічної мови), який може розширювати поведінку CMS, а саме: здатність: (1) перехопити певні події, що виникають при веб-запиті; (2) впровадити функціональні можливості для дій, визначених у відповідній моделі CMS-ML.

Враховуючи відповіді, отримані в попередніх пунктах, необхідно визначити відношення композиції:

1. Між Компонентом та його HTML-частиною (при визначенні нових компонентів);
2. Між веб-сайтом, його сторінками та їх компонентами;

3. Між фрагментом вихідного коду та сутністю, що містить цей код (як правило, веб-сайт, сторінка або компонент).

Враховуючи ці відносини, мова CMS-IL вимагає наявності щонайменше двох метарівнів (крім метарівня, який буде містити сутності CMS): один метарівень - для CMS розширень та компонентів визначених користувачами, а інший - для визначення веб-сайту (який буде включати ці компоненти).

На відміну від CMS-ML, яка представлена для цільової аудиторії бізнес-користувачів, які, як правило, використовують простіші мови візуального моделювання, цільова аудиторія CMS-IL - технічні користувачі, які як правило, віддають перевагу текстовим мовам, оскільки їх діяльність часто включає використання текстових мов програмування .

4.2.2 Типи моделей та ролі моделювання

В цілому, процес моделювання CMS-IL дуже схожий на CMS-ML, але він має деякі відмінності в конкретних стадіях процесу. Моделювання CMS-IL в основному фокусується на трьох різних типах моделей:

1. Шаблони веб-сайтів;
2. Анотації веб-сайтів;
3. Набори інструментів.

Ці моделі мають ті самі назви, що і в CMS-ML, оскільки вони виконують однакові ролі, основна відмінність яких лежить в обсязі та деталях цих моделей.

Звичайно, як і в CMS-ML, не потрібно, щоб один з користувачів CMS-IL мав навички для створення всіх типів моделей CMS-IL. Таким чином, ми вважаємо, що розробка моделей CMS-IL, як правило, виконується відповідно до ролей зображених на рис. 21.

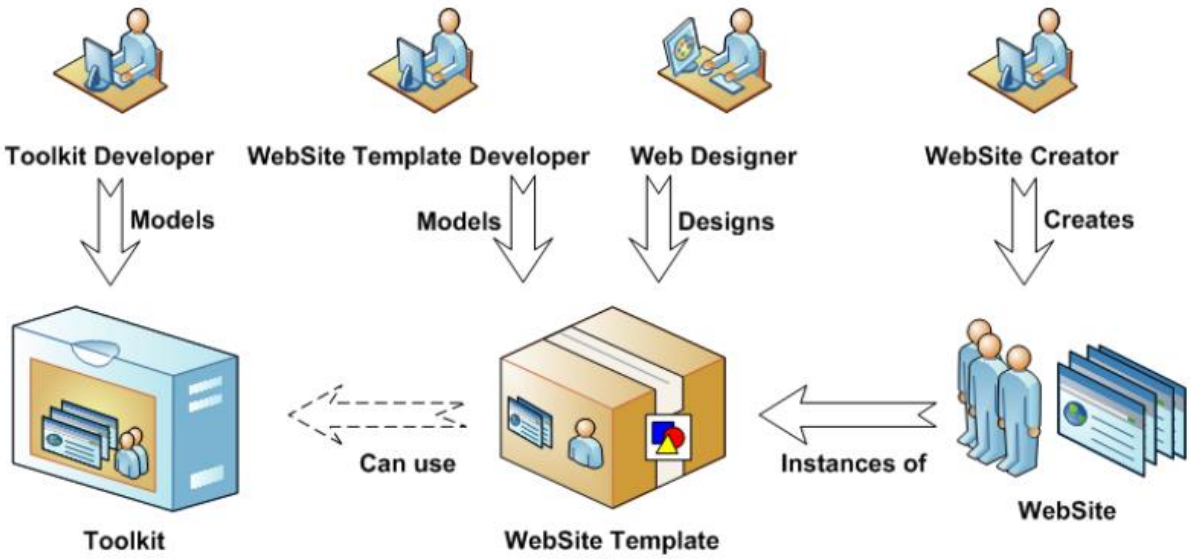

Рисунок - 21 Ролі моделювання в CMS-IL

4.2.3 Опис моделі шаблону веб-сайта CMS-IL

Модель шаблону веб-сайта CMS-IL може розглядатися як розширена версія шаблону веб-сайта CMS-ML. Ця модель дозволяє розробнику шаблонів налаштовувати структуру та поведінку веб-додатка на основі CMS. Для підтримки цієї моделі мова CMS-IL визначає набір елементів моделювання за допомогою яких розробники можуть визначати шаблони. Як і у випадку з шаблонами CMS-ML, шаблони CMS-IL визначаються відповідно до набору представлень (ілюстрованих на рис. 22), а саме:

1. Представлення структури, яке визначає структуру веб-додатка, а саме його сторінок та компонентів;
2. Представлення ролей, в якому визначаються обов'язки, які веб-програма очікує від своїх користувачів;
3. Представлення дозволів, моделювання того, які дії доступні ролям веб-додатка;
4. Представлення Користувачів, яке визначає користувачів, які є важливими для модельованого веб-додатка і повинні бути доступні відразу після розгортання моделі в CMS;

5. Представлення мов, яке стосується локалізації та мов, які веб-додаток повинен мати;
6. Представлення об‘єктів, де розробник може визначити набір об‘єктів (рядків і файлів), які будуть доступні в веб-додатку;
7. Представлення вмісту, яке визначає вміст, який відображатиметься в структурних елементах веб-додатка;
8. Представлення візуальних тем, яке визначає графічний макет і властивості різних структурних елементів веб-додатка.

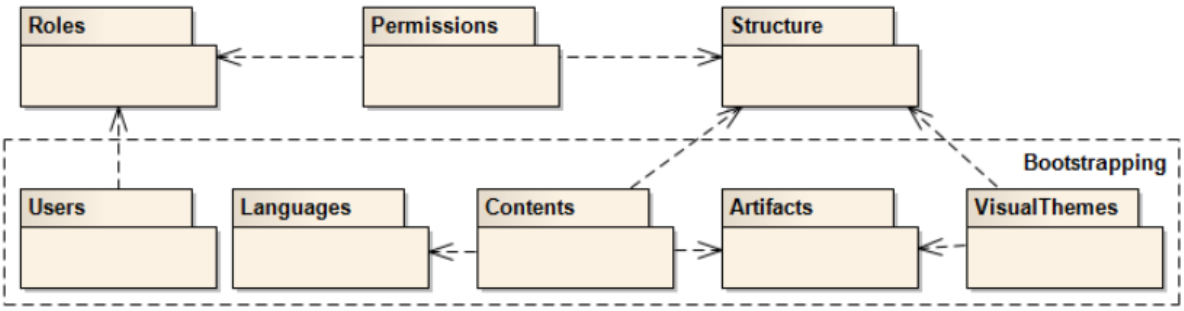

Рисунок - 22 Структура шаблону веб-додатка

Представлення структури є одним з основних елементів шаблону веб-додатка. Це представлення визначає сукупність понять – веб-сайт, динамічна веб-сторінка, контейнер і веб-компонент, які можна вважати практично еквівалентними до структури CMS-ML (оскільки вони мають ті самі призначення). На рис. 23 предемонстрований абстрактний синтаксис для представлення структури. Слід зазначити, відмінності між представленнями структури CMS-ML та CMS-IL, а саме:

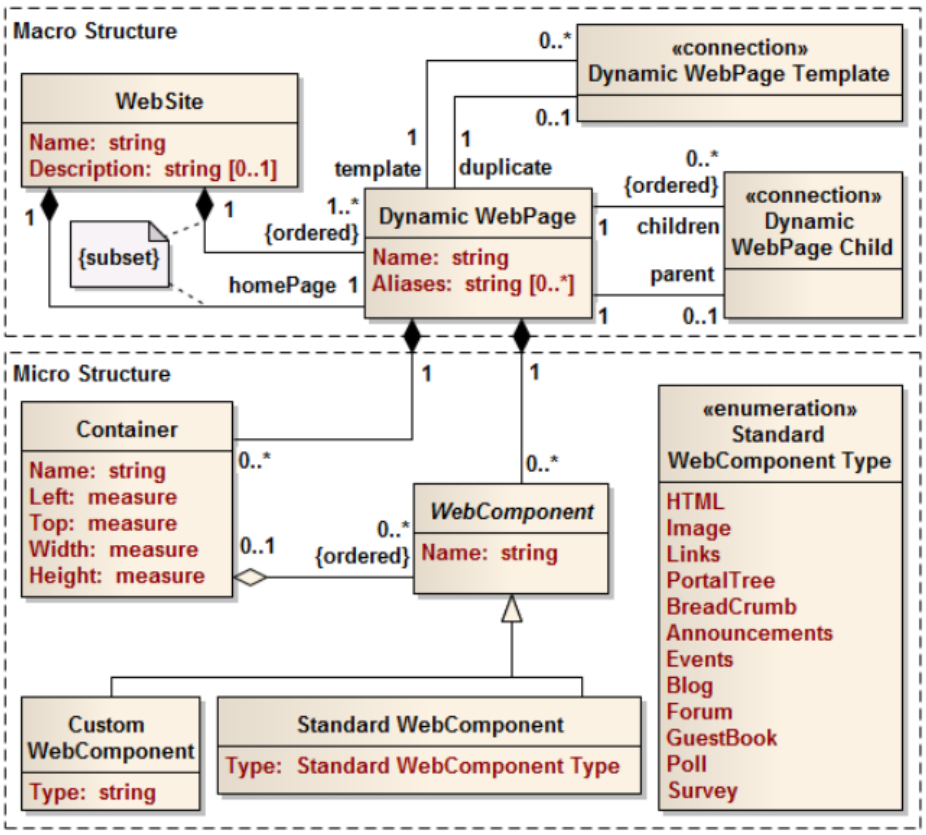

Рисунок - 23 Абстрактний синтаксис представлення структури шаблону веб-сайту

1. WebSite визначає додаткову властивість - опис (Description), яка є необов'язковою та надає опис веб-додатка. Хоча цю властивість можна включити в CMS-ML, більшість користувачів зазвичай вважають її непотрібною, що робить її включення недоцільним. Проте слід зазначити, що ця властивість є важливою, тому що більшість веб-пошукових машин використовують її при відображенні результатів пошуку;
2. Динамічна веб-сторінка надає властивість псевдонімів, що складається з набору рядків, які визначають альтернативні назви для сторінки.;
3. Концепції WebSite, Dynamic WebPage, WebComponent та Container успадковуються від іншого абстрактного поняття - структурного елемента.

Представлення ролей, ідентичне представленню ролей у CMS-ML, описує види відповідальностей, які будуть мати користувачі веб-сайта на основі CMS. Подібно до CMS-ML, він визначає два поняття: роль та делегація ролі, які можуть бути використані для моделювання відповідальностей та визначення, чи можуть вони також бути виконані іншими ролями, відповідно. Рис. 24 ілюструє абстрактний синтаксис для представлення ролей.

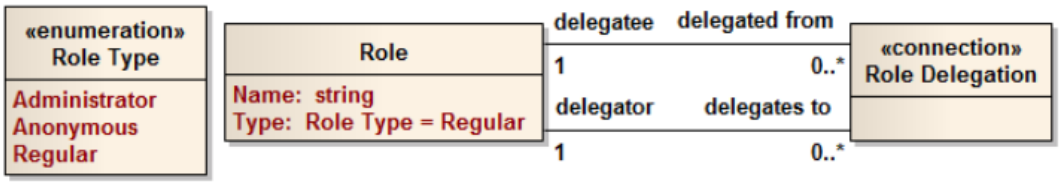

Рисунок - 24 Абстрактний синтаксис представлення ролей

Представлення дозволів, яке дуже схоже на його відповідне представлення CMS-ML, відповідає за встановлення відповідності між ролями та структурними елементами веб-сайту, а саме, його динамічними веб-сторінками та веб-компонентами. Воно визначає такі два поняття, як дозвіл динамічної веб-сторінки і дозвіл компонента. Ці поняття мають такі самі назви, що й у представленні дозволів CMS-ML, оскільки вони мають однакову функцію: створення відносин роль-сторінка та роль-компонент. На рис. 25 наведено ілюстрацію цих понять.

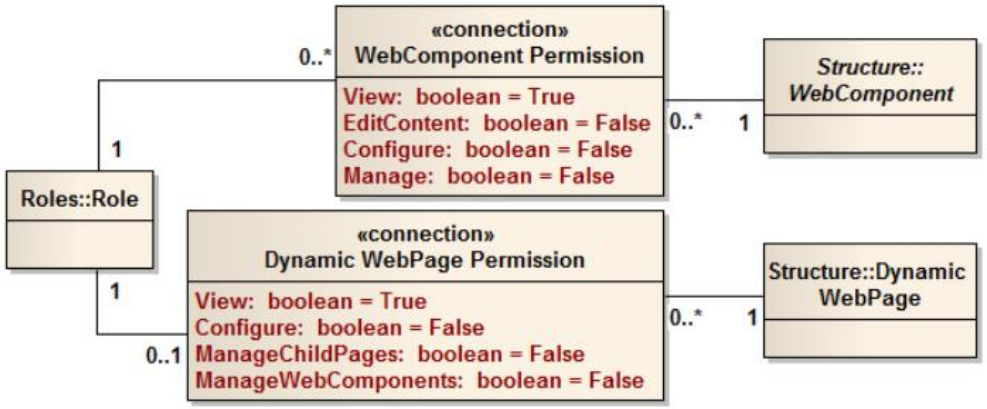

Рисунок - 25 Абстрактний синтаксис представлення дозволів

Представлення користувачів відповідає за ідентифікацію всіх користувачів CMS, які повинні бути доступними кожного разу, коли розгортається модель CMS-IL, а також за їх ролі. Цей представлення визначає дві концепції: користувач та призначення. «Користувач» визначає користувача CMS (особу, яка буде взаємодіяти з модельованою веб-програмою). «Призначення» моделює розподіл між користувачем та роллю - тобто очікувані обов'язки в межах веб-додатка, до якого призначено користувача. Рис. 26 ілюструє абстрактний синтаксис для представлення користувачів.

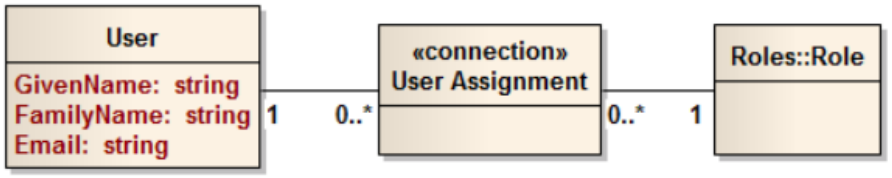

Рисунок - 26 Абстрактний синтаксис для представлення користувачів

Представлення візуальних тем призначене для ролі веб-дизайнера. Воно визначає, як користувачі будуть бачити структурні елементи веб-програми з точки зору візуальних елементів, таких як колір, ширина або товщина лінії. Це робиться шляхом визначення класів CSS (Cascading StyleSheet) та вбудованих стилів відповідно до найкращих практик веб-дизайну. Абстрактний синтаксис цього представлення зображений на рис. 27.

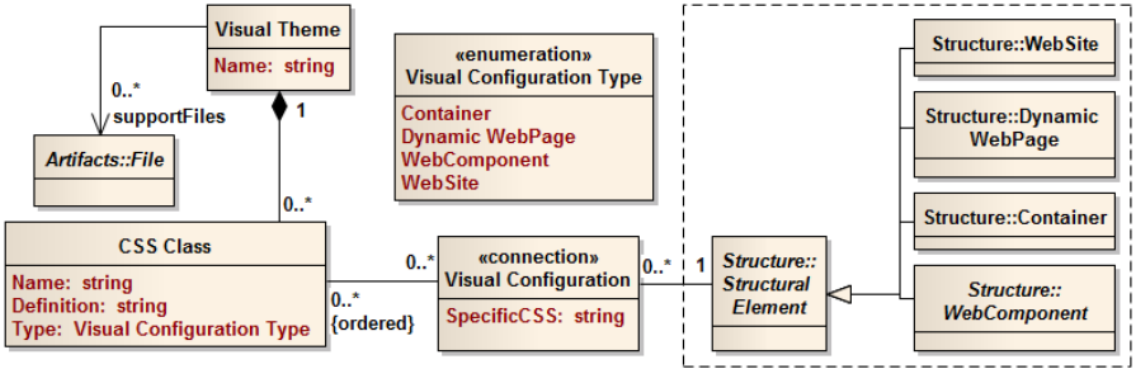

Рисунок - 27 Абстрактний синтаксис представлення візуальних тем

4.3 Засіб синхронізації

Мови CMS-ML та CMS-IL є невід'ємними компонентами рішення, запропонованого в розділі 3. Однак, хоча ці мови можуть відповідати вимогам моделювання їх цільових аудиторій для створення нових веб-додатків на основі CMS, все ще існує потреба в механізмі, який:

1. Підтримує отримання моделей CMS-IL з моделей CMS-M;
2. Забезпечує зберігання обох типів моделі у відповідності один з одним.

Відсутність такого механізму означала б, що моделі на різних мовах (наприклад, CMS-ML та CMS-IL) повинні бути синхронізовані вручну, що матиме наступні недоліки: (1) потребу в додаткових зусиллях для виконання ручної синхронізації ; (2) збільшення кількості помилок через необхідність виконання повторюваних дій.

Мова для синхронізації моделі MYNK (англ. Model sYNchronization frameworK), повинна бути створена для вирішення цієї потреби. MYNK дозволяє отримати модель CMS-IL з моделі CMS-ML (і навпаки), а також гарантувати, що зміни до однієї з моделей - або навіть до обох моделей - будуть поширюватися на інші моделі, щоб підтримувати узгодженість між ними. У цьому підрозділі ми коротко опишемо MYNK. Зокрема, в цьому розділі описується: (1) обґрунтування визначення мови MYNK; (2) підхід, який MYNK використовує для підтримки моделей, які відповідають один одному; (3) деякі аспекти вирішення конфліктів, а також сумісність між MYNK та мовами моделювання.

4.3.1 Сучасні проблеми трансформацій моделей

Незважаючи на те, що існують моделі трансформації мов та структур, ми не знайшли жодної, яка ефективно підтримує сценарії, в яких як джерело, так і цільові моделі зазнають змін, які роблять їх несумісними між собою.

Слід зазначити, що існує явна відмінність між трансформацією моделі та синхронізацією моделі. Трансформація моделі отримує вихідну модель S і виводить цільову модель O; також для трансформації моделі можливо отримати вихідну модель S та цільову модель T і змінити T, так щоб вона стала еквівалентом S. З іншого боку, синхронізація моделі отримує дві моделі A і B і змінює обидві моделі так, щоб вони стають еквівалентними одна одній (якщо можливо).

Крім того, трансформацію моделей можна розглядати як особливий випадок синхронізації моделей. Типові перетворення моделі, які отримують вихідну модель S1 і виводять іншу модель O1, - визначені відповідно мовами S та O, - на практиці еквівалентні до (1), визначення нової порожньої цільової моделі O0, і (2) зміна O0 до стану еквівалентного S1, і, таким чином, отримання іншої моделі O1. Навіть перетворення QVT (які отримують джерело та цільову модель і можуть відповідно змінювати цільову модель) працюють як механізм синхронізації моделей, оскільки вони (1) аналізують вихідну модель та цільову модель, (2) визначають невідповідності між ними, і (3) якщо вони можуть змінити цільову модель, змінюють (або видаляють) ці невідповідності.

Хоча аналізовані мови можуть підтримувати модифікацію моделі B1, щоб стати еквівалентною моделі A, це, як правило, означає, що деякі зміни, внесені до B1 (особливо зміни елементів, які також були змінені в A) будуть втрачені. Рис. 28 наочно показує цю проблему.

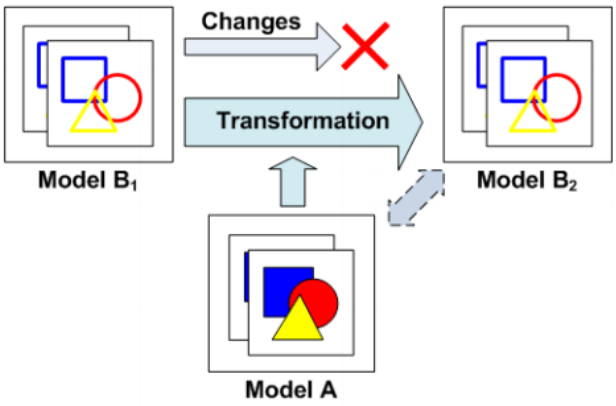

Рисунок - 28 Втрата інформаці при трансформації моделей

Це є особливо важливим обмеженням для запропонованого підходу розробки веб-додатків (раніше описаного в розділі 3), в якому передбачається, що і бізнес-конструктор, і системний конструктор (ролі, передбачені цим підходом) можуть одночасно внести зміни до їх відповідних моделей. Очевидно, було б неприпустимо, щоб явні зміни, внесені будь-якою роллю, втрачалися при перекладі.

У контексті запропонованого підходу розробки синхронізація моделей вирішує такі завдання: (1) забезпечення того, щоб моделі CMS-ML та CMS-IL були семантично еквівалентні одна одній; (2) забезпечення того, щоб ця семантична еквівалентність зберігалася протягом змін моделей, незалежно від того, які саме моделі змінювались.

Крім того, синхронізація моделей вважає модель не просто спрощеним представленням певної реальності в її останньому стані, а скоріше сумою її історії (тобто усіх змін, які вона зазнала, поки вона не дійшла до поточного стану).

Важливо зауважити, що синхронізація моделей не призначена для забезпечення інтерактивних сценаріїв в зворотному напрямку, огляд яких наведено на рис. 29: (1) розглядаючи модель A1 (зазначену певною мовою A), модель B1 отримана за допомогою якогось механізму (наприклад, трансформації моделей T(A-B)); (2) аналогічно, використовуючи

трансформації моделей T(B-A), модель A2 отримується з B1. Якщо перетворення T(A-B) і T(B-A) є точно симетричними (тобто вони є протилежними один одному), то A1 і A2 повинні бути семантично еквівалентними (у тому сенсі, що вони повинні мати однакове значення). Однак, даний сценарій має необхідність того, щоб обидві мови мали однаковий рівень експресивності, оскільки в іншому випадку інформація між різними перетвореннями буде втрачена.

Замість цього, синхронізація моделей пропонує сценарій, як показано на рис. 30. Цей сценарій починається з двох семантично еквівалентних моделей ML1 та IL1, які моделюються відповідно мовами CMS-ML та CMS-IL. Після того, як бізнес-конструктор вносить зміни (позначено CML1 на малюнку) до ML1, для того, щоб отримати іншу модель ML2, механізм синхронізації моделі отримає ці зміни та генерує відповідний набір змін (позначений CIL1) на мові CMS-IL. Після цього CIL1 застосовується до IL1, створюючи іншу модель IL2, яка має бути семантично еквівалентною ML2.

Проте на практиці невідомо, чи моделі CMS-ML та CMS-IL будуть синхронізовані після простого поширення змін (насправді, це малоймовірно). Це пов'язано з тим, що можуть бути внесені нові зміни до моделі CMS-ML, на основі (1) змін, внесених до моделі CMS-IL (конструктором системи), або (2) змін, які передбачаються початковою операцією поширення змін (показано на рис. 30). Більш конкретно, операція синхронізації моделі повинна складатися з циклу CMS-IL → CMS-ML ⇒ CMS-ML → CMS-IL, який повинен бути виконаний до досягнення фіксованої точки.

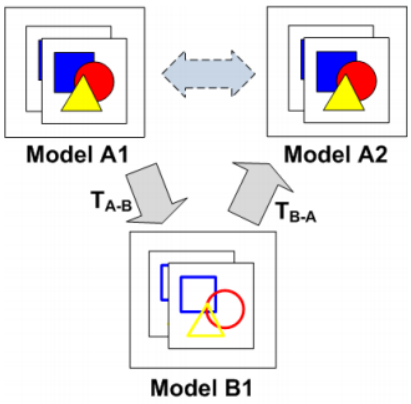

Рисунок - 29 Інженерія в зворотньому напрямку

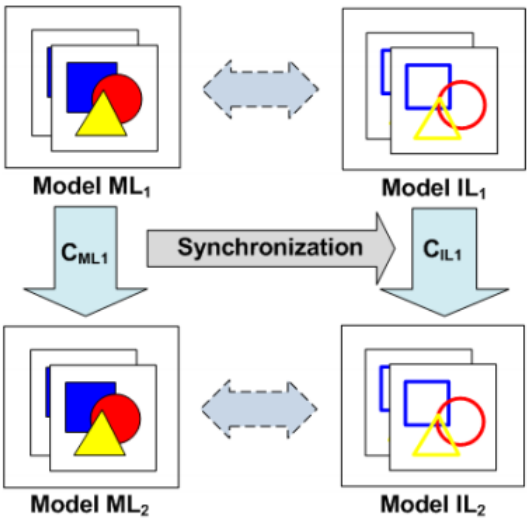

Рисунок - 30 Синхронізація змін моделей

У даному розділі було представлено рекомендації та пропозиції щодо створення мов моделювання та засобу синхронізації моделей. На прикладі опису абстрактного синтаксису одного з типів моделей даних мов (шаблона веб-сайта) показано приклад застосування підходу до розробки, розглянутого у розділі 3.

ВИСНОВКИ

Інтернет вплинув на концепцію розробки та розгортання більшості додатків. Розробники тепер створюють веб-додатки, і для цього потрібно мати справу з різними поняттями, такими як запити, файли «cookie» та гіперпосилання.

Особливий вид веб-додатків, що набуває популярності сьогодні - системи керування вмістом (англ. CMS). Зазвичай такі програми надають значну ступінь налаштовування та розширюваності, яка зменшує необхідність розробки нових програмних систем для підтримки розподілу вмісту та спрощує виконання багатьох користувацьких задач.

Розроблена система, яка пропонує загальні варіанти вирішення проблем, що виникають при розробці веб-додатків, а також запропоновані розширення, які полягають у розробці компонентів або модулів, що будуть встановлені поверх системи CMS.

Запропоновані модель-орієнтовані підходи (англ. MDE), які спрямовані на полегшення процесу розробки, зосереджуючись на моделях та концепціях замість вихідного коду. Парадигма MDE спрямована на досягнення більш високого рівня абстракції, виступаючи за те, що моделі повинні бути основними артефактами у процесі розробки програмного забезпечення, тоді як інші об‘єкти (наприклад, вихідний код та документація) можуть бути отримані з цих моделей у автоматичному режимі, використовуючи перетворення моделей.

Системи CMS мають потенціал для того, щоб стати наступним стандартом фреймворків для веб-додатків. Вони також можуть скористатися перевагами, наданими парадигмою MDE, оскільки ці підходи можуть значно прискорити розробку та впровадження веб-додатків та компонентів, а також спростити їх технічне обслуговування.

Запропоновано новий підхід, орієнтованй на MDE, для розробки CMS веб-додатків. Цей підхід зосереджено на розробці веб-додатків, що базуються (і розгортаються) на системах CMS, і відрізняються від інших через використання кількох мов моделювання, а також використання механізму синхронізації моделі, для забезпечення відповідності усіх представлень системи. Запропонований підхід базується на двох мовах, орієнтованих на CMS, CMS-ML та CMS-IL, які знаходяться на різних рівнях абстракції. CMS-ML розглядає концепції моделювання на високому рівні, в той час як CMS-IL використовує концепції низького рівня, а також забезпечує спільну основу для побудови мов більш високого рівня, таких як CMS-ML.

Запропонований підхід забезпечує зменшення додаткової роботи, необхідної для вирішення проблеми невідповідності між представленням системи кінцевими користувачами та розумінням розробниками цього представлення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Model-Driven Software Development: Technology, Engineering, Management. / [Völter M., Stahl T., Bettin J., Haa A.] - New Jersey, U.S.A.: John Wiley & Sons,Hoboken, 2005.
2. Schmidt C. Guest Editor’s Introduction: Model-Driven Engineering. IEEE Computer, 2006.
3. Professional C# 4.0 and.NET 4. / [Nagel C., Evjen B., Glynn J., Watson K., Skinner M.] - Wrox, 2010.
4. Schildt H. Java The Complete Reference. McGraw-Hill Osborne Media, 2011.
5. Programming PHP. / Lerdorf R., Tatroe K., MacIntyre P. O’Reilly Media, 2006.
6. Content Management Systems (Tools of the Trade). / [Suh P., Addey D., Thiemecke D., Ellis J.] - Glasshaus, 2003.
7. Boiko B. Content Management Bible. New Jersey, U.S.A.: John Wiley & Sons, Hoboken, 2001.
8. Rockley A. Managing Enterprise Content: A Unified Content Strategy. New Riders Press, 2002.
9. MDA Explained: The Model Driven Architecture: Practice and Promise. / Kleppe A., Warmer J., Bast W. - Massachusets, U.S.A: Addison-Wesley, Reading, 2003.
10. Kelly S. Domain-Specific Modeling: Enabling Full Code Generation. / Kelly S., Tolvanen J.P. - New Jersey, U.S.A.: John Wiley & Sons, Hoboken, 2008.
11. Ткаченко Р.Ю. Теоретичні та прикладні аспекти розробки комп’ютерних систем. Технології проектування і розробки інформаційних систем. Моделювання в науці та інженерії. 2018.
12. Stachowiak H. Allgemeine Modelltheorie. Springer-Verlag. 1973.
13. Atkinson C. Reducing accidental complexity in domain models. / Atkinson C., Kühne T. - Software and Systems Modeling. 2008.

14. Guest Editors' Introduction: Model-Driven Development. / Mellor J., Clark N., Futagami T. - IEEE Software, 2003.

15. Favre J.M. Language Engineering for Model-Driven Software Development. Dagstuhl, Germany. 2004.

16. Thomas D.A. MDA: Revenge of the Modelers or UML Utopia? IEEE Software, 2004.

17. Henderson-Sellers B. UML – the Good, the Bad or the Ugly? Software and Systems Modeling, 2005.

18. Model Driven Engineering Languages and Systems. / [France R.B., Kazmeier J., Breu R., Atkinson C.] - Springer Berlin/Heidelberg, 2005.

19. Web Engineering: Modelling and Implementing Web Applications. / Brambilla M., Comai S., Fraternali P. - London. 2007.

20. Kroiß C. UWE Metamodel and Profile: User Guide and Reference. / Kroiß C., Koch N. - Ludwig-Maximilians Universit¨at, 2008.

21. Severdia R. Using Joomla: Building Powerful and Efficient Web Sites. / Severdia R., Crowder K. - O'Reilly Media, 2009.

22. Kelly S. Worst Practices for Domain-Specific Modeling. / Kelly S., Pohjonen R. - IEEE Software, 2009.

23. Standish A. American Federation of Information Processing Societies: Proceedings of the 1975 National Computer Conference // Extensibility in programming language design. New York, NY, USA. 1975. pp. 287–290.

24. Абельсон Г. Структура та інтерпретація комп'ютерних програм. / Абельсон Г., Сассман Д. - MIT Press, 1996.

25. Atkinson C. Profiles in a Strict Metamodeling Framework. / Atkinson C., Kühne T. - Science of Computer Programming, 2002.

26. Are Domain-Specific Models Easier to Maintain Than UML Models? / Cao L., Balasubramaniam R., Matt R. - IEEE Computer, 2009.

27. Debasish G. DSL for the Uninitiated. 2011.

28. Atkinson C. Profiles in a Strict Metamodeling Framework. / Atkinson C., Kühne T. - Science of Computer Programming. 2002.

29. Vengette Corporation. The Web Content Battle: Managing Pages Vs. Content – Which One Is Better For You? 2009.